DOCUMENT RESUME

ED 126 923 IR 003 838

AUTHOR Swallow, Ronald J.
TITLE CHARGE Interactive Graphics System Terminal: Theory of Operation. Technical Report 74-26.
INSTITUTION Human Resources Research Organization, Alexandria, Va.
SPONS AGENCY Army Research Inst. for the Behavioral and Social Sciences, Arlington, Va.
REPORT NO HumRRO-TR-74-26
PUB DATE Dec 74
CONTRACT DAHC-19-73-C-0004
NOTE 72p.; Technical Report

EDRS PRICE MF-$0.83 HC-$3.50 Plus Postage.
DESCRIPTORS *Computer Graphics; Computer Programs; *Computers; *Display Systems; Electronic Equipment; Information Processing; *Input Output Devices; Instructional Media; *On Line Systems; Screens (Displays); Technological Advancement; Three Dimensional Aids; Visual Aids
IDENTIFIERS CHARGE; Computer Terminals; HumRRO; Image Encoding; Interactive Computer Systems

ABSTRACT

The CHARGE computer terminal can provide graphics display for many applications in color, gray-level, 3-D, perspectives, and rapid updating. Perspective views can be generated from a three-dimensional coordinate system which changes to match actual physical descriptions. Image encoding and hardware design are described from a theoretical and detailed logic point of view. (CH)

Technical
Report
74-26

HumRRO-TR-74-26

HumRRO

CHARGE
Interactive Graphics System Terminal:
Theory of Operation

Ronald J. Swallow

HUMAN RESOURCES RESEARCH ORGANIZATION
300 North Washington Street • Alexandria, Virginia 22314

Approved for public release, distribution unlimited.

December 1974

Prepared for

U.S. Army Research Institute for the
Behavioral and Social Sciences
1300 Wilson Boulevard
Arlington, Virginia 22209

Recognizing the importance of computer-administered instruction in meeting the Army's training needs, HumRRO conducted a system design study in 1971 and 1972 that led to the development of an interactive graphics system called CHARGE. In designing the CHARGE terminal, two problems had to be solved—how to code and buffer a high-resolution colored image, and how to decode and display the image on a CRT, both at a reasonable cost. The report describes the terminal's image encoding hardware architecture from a theoretical and detailed logic point of view.

Designed initially for use in instructional systems, the CHARGE terminal can provide considerable and versatile graphics display for many applications (including equipment design, trouble-shooting, etc.). While being cost-competitive, the CHARGE terminal system can achieve color, gray level, 3-D perspectives, and rapid updating. Perspective views can be simply, rapidly, and economically generated from a (3-D) xyz-coordinate, dynamically changing, "real world" description.

PREFACE

This report describes the results of the development by the Human Resources Research Organization (HumRRO) of the CHARGE Interactive Graphics System terminal. The terminal's image encoding and terminal architecture are described from a theoretical and detailed logic point of view.

The work was begun, at HumRRO Division No. 1 (Systems Operations), Alexandria, Virginia, under Project IMPACT in 1971 and has continued under Work Unit CATALIST. Dr. J. Daniel Lyons is Director of the Division (now the HumRRO Eastern Division), and Dr. Robert J. Seidel is Program Director, Instructional Technology Group.

Members of the CHARGE System R & D Team were Dr. Ronald J. Swallow, Principal Investigator, Mr. Roger L. Gunwaldsen, and Mr. William G. Underhill.

The work described in this report was conducted for the Department of the Army under Contract 19-73-C-0004. Computer-administered instruction research is conducted under Army Project 2Q163101A734.

Meredith P. Crawford
President
Human Resources Research Organization

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

INTRODUCTION

BACKGROUND

The combination of shrinking financial resources and a smaller, volunteer Army increases the demands made on service personnel, increases the importance of each individual soldier, and poses even more difficult problems in training. The need for more effective and efficient training is increasing. The training must deal with widespread student differences, provide an increasing number of complex skills, and require fewer skilled instructors.

When it is developed properly, computer-administered instruction (CAI) is one of the most promising approaches available to meet these new training demands. In recognition of these demands, the Army Research Office established Project IMPACT in FY 1968 as an advanced development effort to provide one means to achieve these goals. A complementary effort to study feasibility of CAI in an operational environment was established at Fort Monmouth, New Jersey. The Army's six years of accumulated experience in CAI research and development have resulted in a ready, advanced development resource for the Army's needs.

HumRRO's recent capital investment in a mini-computer CAI configuration has added new dimensions to the instructional facilities. The Project IMPACT staff has provided guidance including simulations leading to the design of the Army's prototype Computerized Training System (CTS). Work is continuing under Work Unit CATALIST.

The overall system research and development efforts, of which the CTS simulation are one output, have been intended to develop a new-generation, dynamic-graphics, cost-competitive CAI system. The current HumRRO-designed architecture represents a truly state-of-the-art system.

HumRRO's system design study conducted during 1971 and 1972 led to a hardware/software design which in the area of terminal systems exceeded expectations. While being cost-competitive, it achieves not only color, gray level, 3-D perspective, and rapid updating, but perspective views can be simply, rapidly, and economically generated from a (3-D) xyz-coordinate, dynamically changing, "real world" description.

The capability is provided in a video display terminal called CHARGE. Designed initially for use in instructional systems, the CHARGE terminal can provide considerable

and versatile graphics display for many applications (including equipment design, trouble-shooting, etc.)

The HumRRO design study indicated that, with respect to leading existing CAI system designs (e.g., PLATO and TICCIT), an improvement of four to 10 in instructions executed/user/second, and an improvement on the order of a factor of six in lesson size and swap size were feasible. These advances are possible in the CHARGE Interactive Graphics System at no increase in user cost over PLATO or TICCIT and with no significant reductions in any other system parameters. Also, the proposed design is economical for a system with as few as 100 terminals, yet modular for expansion beyong 1000 terminals without duplication of text storage-retrieval subsystems. All text material is centralized, with little or no need for films or visual materials at the terminals.

Essential elements in achieving this advanced design are summarized as follows:

(1) Terminal architectures that incorporate new solid-state devices, that is, CHARGE (Color Halftone Area Graphic Environment) terminal.

(2) Special-purpose hardware to take over well-defined and stable software functions (i.e., image generator for graphics transformations from 3-D to 2-D).

(3) Elimination of Input/Output bottlenecks within the central computer system by using high-speed drum swap and building a few special interfaces where necessary.

(4) The latest computer Central Processing Unit (CPU) and Random Access Memory (RAM) components for mini- and midi-computers where more production cost-effectiveness can be realized.

The power of the special-purpose hardware produces cost-performance gains that are orders of magnitude above what is possible through the use of a general-purpose computer and software. Once the special-purpose hardware has been incorporated into a system, it is a simple matter to reconfigure the design to lower performances, thereby achieving significant reduction in cost over existing CAI designs; this can be done only with great difficulty using the reverse approach, that is, by first designing the simpler one and later modularizing upward. Thus, the entire range of users' needs, which vary from one end of the spectrum to the other, can be met economically by the new design through reconfigurations of its more efficient components.

The resolution, response time, and color sensitivity of the human eye form the boundary of display terminal performance. Off-the-shelf terminals have fallen short not only of this boundary, but also of the performance of known technology. The CHARGE

Interactive Graphics System closes the gap between feasibility and availability. Its terminals approach the design limit, but do so without accompanying high costs.

The CHARGE display terminal system also includes an image generator to support its terminals. Because of this special-purpose hardware, each CRT can act as a "window into a time-varying world," a world defined physically in x, y, z, and t rather than as a canned sequence of two-dimension perspectives, a world whose perspectives contain vivid, colored surfaces rather than lines alone.

Thus, not only is the human eye's perception capability matched, but so is the human mind with its capability to model and manipulate real or abstract worlds in *real time.*

The purpose of this report is to describe the terminal portion of the CHARGE Interactive Graphics System. The CHARGE image generator and the CHARGE software support are described in other documents.

SIMULATION OF 3-D WORLDS

The terminal in the CHARGE Interactive Graphics System acts as a window into a 3-D world, a world that is stored in the computer as it is (i.e., as it is defined) not how it looks from a perspective. This world can be modeled closely to that of a 3-D world, or as a symbolic or abstract world. The objects of the world can have dynamic parameters, such as velocity and acceleration. Objects can be programmed to obey physical laws that include collision dynamics, where an object bounces according to the conservation of momentum and energy.

The objects of a world as observed at a terminal can actually simulate or represent other users of the system. Thus, many users of terminals can observe the same subworld, simultaneously, each seeing the other as if he were really there. This permits, for instance, war games. It also makes possible driver training in which each driver finds himself in a real-world situation with others involved, producing the dangers and the problems associated with learning to drive. It can act as a pilot trainer where other planes controlled by other terminals are within the same world and act as realistic obstacles within the learning environment.

Via a "joy stick" or keyboard, the user may inquire within a subworld as if he were there He can alter the position of objects by commands to the keyboard or joy stick; he can alter his own position and orientation within the world, which includes velocity and

acceleration. A designer can construct or take apart an object, and peer from any point within the object. In general, the designer has all the degrees of freedom conceivable, but prior to the requirement for expensive and relatively unwieldy mock-up stage of model construction. With one key press, a user can have an object added to his visible world. A new perspective can be calculated and transmitted back to his display within .15 second.

For example, a truck can be specified down to its last bolt, defining a subworld in which a student can roam. Within this subworld, via support of the image-generation hardware and via joy stick that controls his position within the environment, he can open a door, remove a wheel, or operate the truck since dynamic parameters such as movement and acceleration can be controlled. Pressing the accelerator can be made to increase the rate of rotation of such components of the engine as the crankshaft, differential, and wheels. Each of these subcomponents of the image can be made visible, while the remaining components (the frame, seats, etc.) can be made invisible. The user (designer, troubleshooter, student, etc.) can zoom to get a clear view of what other part he so desires, amplified to as great detail as available in that part's definition.

Similarly, the anatomy of a human being can be described down to the last organ. A medical student may then dissect, as he so desires, by making invisible those portions necessary in order to view the components he wishes to look at. The dynamics of blood flow can be implemented, as well as the heartbeat and the lung expansion and compression. All of this will be in vivid color, with curved surfaces appearing smooth, not as small facets as on a diamond.

AUTHORING

From an author point of view, worlds are defined in many different ways and at various levels of complexity. For instance, objects already defined by other authors are retrievable by each author by name. The author who needs a car can call it by that name, position it where he wishes, re-color it if he so desires, and add it to a subworld to create the situation desired for a learning situation. If an object does not exist, an author can rely upon software support close to that needed by an architect to create the object. Here the author is working at a lower level, having to either mold an object into the desired shape, form it by rule, assemble it out of basic objects, or construct it by specifying points on the surface of the object.

Each world, or subworld, defined by an author can be given a name and retrieved later by input of the name only. A subworld itself, called for by name, can be treated as an object and used to create a larger world. The language will provide for commands (such as "Put the lamp on the table," "Move the table into the corner.") The author language is being designed to be expandable to include new commands as authors so require. This is accomplished by coding author commands in a language of a higher level than machine language.

SYSTEM OVERVIEW

The CHARGE system has three major components: a host central computing system, an image generator, and a set of user display terminals. These three components are configured as in Figure 1.

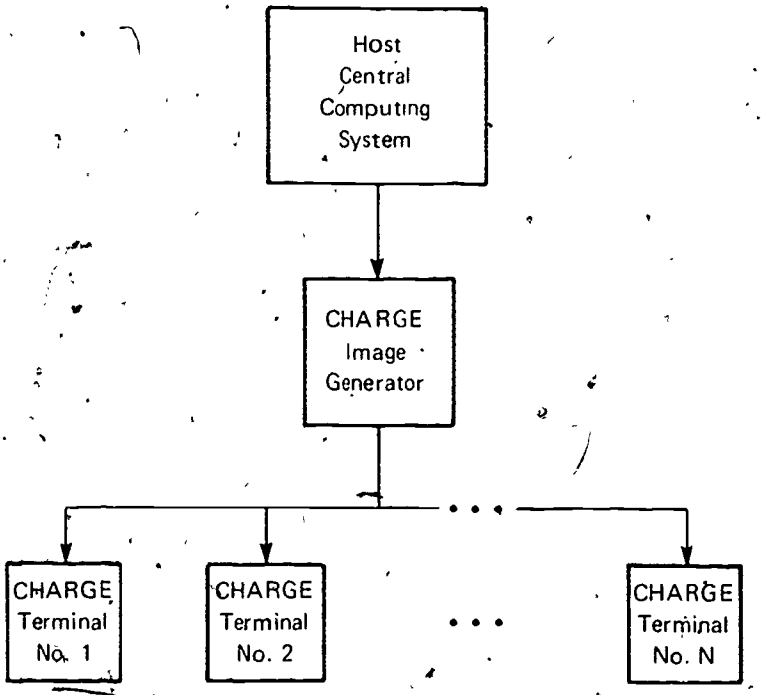


Figure 1 – The CHARGE System

The host central computing system interacts with each user as he creates his 3-dimensional (3D) "world," a description of a 3-dimensional region of space. The host does this by maintaining and manipulating two separate files that serve as the data base for the CHARGE system:

- The first file is the "atom" file, a low-level data base that defines a library of standard 3-dimensional shapes or building blocks such as a unit cube, unit sphere, and so

forth. These atoms are generally not modified by the user; in fact, the user does not even have to know what data are used to represent an atom, he merely has to know each atom's standard orientation in space.

• The second file is the "object" file, a three-structured file by which the user can create and name complex objects by defining them in terms of previously defined objects and atoms. The user does this by use of symbolic commands such as: "Create Object A by adding Objects B and C. Then create Object D by adding Objects A, B, and A, this time moved up 3 meters in y in the xyz space." The linkages between levels of object definition are "transformations" (such as, "move 3 meters in y"). The lowest level definition in any object is always a transformed atom. The object file structure for Object D in the above example is given in Figure 2.

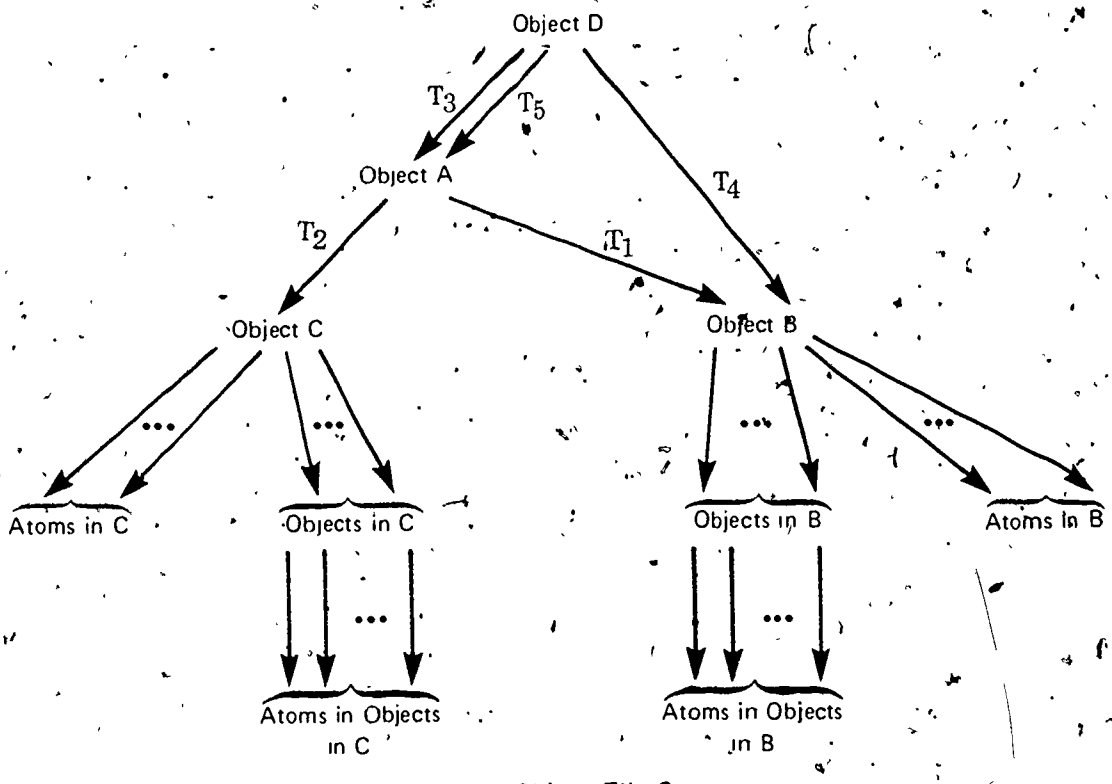


Figure 2 – **Object File Structure**

It should be noted that the data structures in the host central computing system do not represent the graphic view of the world, rather, they represent a data base from which an infinite number of graphical perspectives can be made merely by the command, "Put me at (x,y,z) in this world and show me what I see." In fact, the host can put several users into the same world and generate views for each using the same data base. In this aspect, the host

system is far more efficient in its file storage than it would be if it had to store "canned" (i.e., pre-computed) perspectives. Furthermore, the CHARGE system has the capability of generating a dynamical representation of this 3D world by giving the transformations and/or observer time varying parameters. The cost and performance of the host is shared by the N users.

The process of generating the perspective begins at the time the user gives the "show" command to the host. The host then initiates this process by first "compiling" the user's world. The compilation process accomplishes two tasks:

(1) Each atom that is symbolically referred to in the object file has its atom data retrieved from the atom file.

(2) Each transformation string linking an atom to the world is replaced with an equivalent composite transformation.

These two sets of data now define the user's symbolic world. There is one composite transformation for each occurrence of an atom in the user's symbolic description. The amount of actual atom data is related to the number of different atoms symbolically referred to by the user. For example, if the user's symbolic object file had 50 atoms, each one a Cube, the compiler would produce 50 composite transformations but the only atom data that would be retrieved is that of the atom Cube. These data along with the user's viewing orientation in his 3D world are then sent to the CHARGE image generator for perspective computation thereby freeing the host for another user.

The top half of Figure 3 depicts each user's flow through the host computing system as he creates and views a 3D world. A typical user will spend most of his time in a "think" mode causing the host to loop through all of the "No" branches each time the user is polled. The top three boxes are entered infrequently compared to think time. When they are entered, they are typical interactive user requests requiring very small amounts of the host's resources.

The greatest load to the host computer is that of compiling a user's world, but this too is a low-frequency event. Furthermore, the demands on the host are still quite small. A world with 100 atoms, each with an average linkage depth of 4 levels would require approximately $100 \times 4 \times 27 \doteq 1.1 \times 10^4$ multiplications to produce the 100 composite transformations. If the host has a multiplication time of 3μsec/multiplication, this would be 33 msec. of computation, a fraction of the real time required to retrieve the user's atom and object files from disk storage. The computational load of computing the user's image is off-loaded onto the CHARGE image generator.

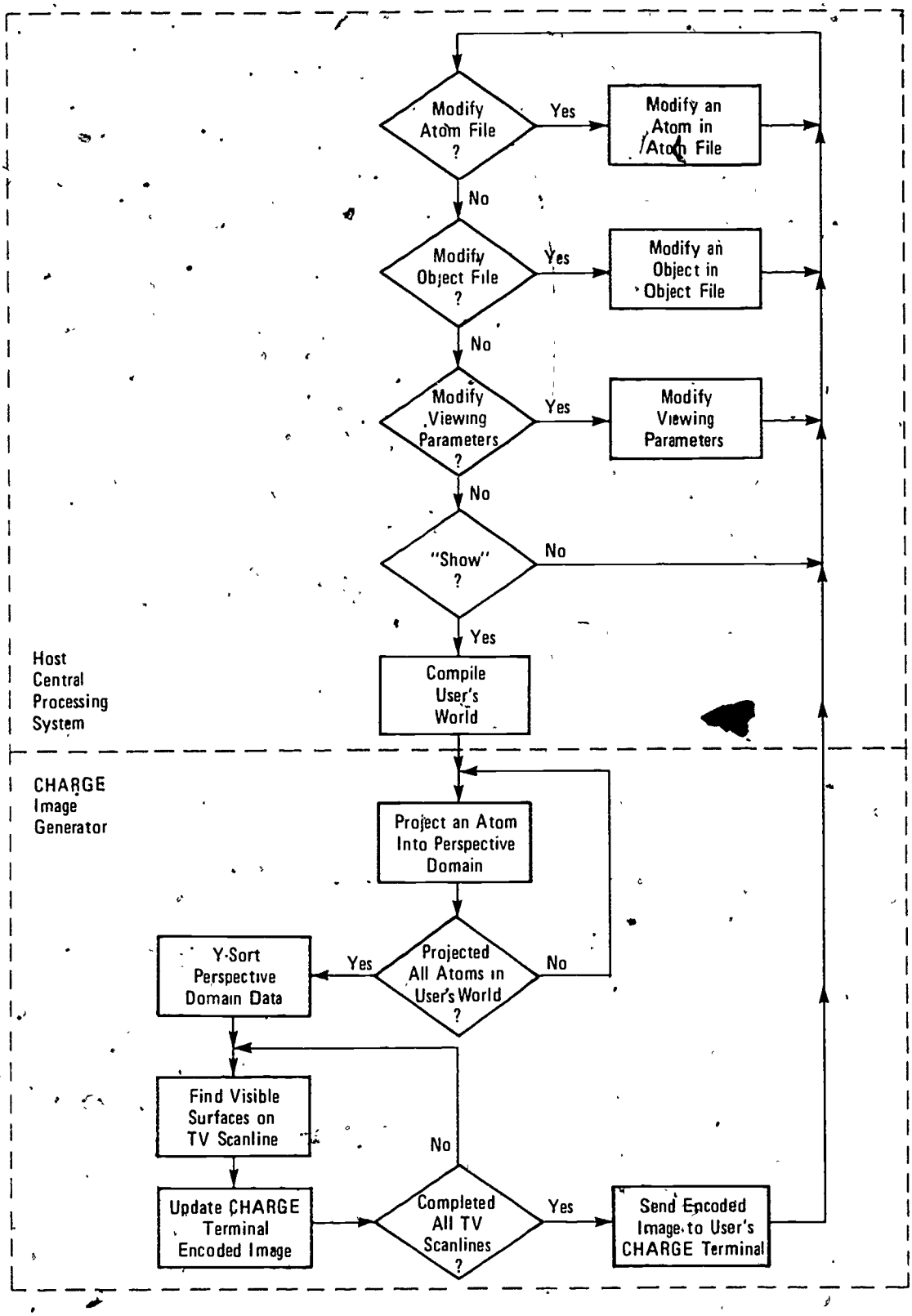


Figure 3 – CHARGE System Functional Flowchart

The CHARGE image generator is a processor located between the host central computing system and the N user terminals. Its function is to translate each user's 3D world/viewer data into a high-resolution encoded image that can be buffered locally at low cost at the user's terminal. Like the host, the image generator's cost and resources are shared by the system's N users. At the present time, the image generator function is being emulated in software, thereby allowing for experimentation in new algorithms before a hardware design is finalized.

The CHARGE image generator is really a two-stage processor. The compiled world data from the host enters the projection processor which projects each atom into a "perspective domain" and limits the resulting perspective domain data to those atom surfaces that would be visible to the user on his monitor if no other atom surfaces were present. The exact form of this perspective domain atom data is chosen to maximize the rate at which the second stage processor, the visible surface processor, is able to determine which surface is actually visible at each (x,y) coordinate on the user's monitor. The visible surface processor also encodes the resultant image into a highly data-compressed format that can be efficiently transmitted to and economically buffered at the user's CHARGE terminal.

Once the encoded image has been transmitted to the user, the image generator is free to service another user. The two stages of the image generator are independent and if a second buffer for y-sorted perspective domain atom data is added, the projection processor can fill one y-sorted buffer with one user's data while the visible surface processor empties the other y-sorted buffer containing a second user's data.

Figure 4 demonstrates the operations of the image generator on an extremely simple case. The user has symbolically created a world consisting of just two atoms: a cube and a triangular plane. The host sends to the image generator the following data:

(1) The user's orientation

(2) Atom data for the standard cube

(3) A transformation orienting the standard cube in the 3D world

(4) Atom data for the standard triangular plane

(5) A transformation orienting the standard triangular plane in the 3D world.

The projection processor first projects the two atoms onto the user's "viewing window," the rectangular area corresponding to the user's monitor. Back surfaces are "clipped away." If any atom surfaces had projected outside of the viewing window rectangle, they too would have been clipped away. The resulting perspective domain data consisting of surfaces and edges is shown in Figure 5. Note that no "front surface"

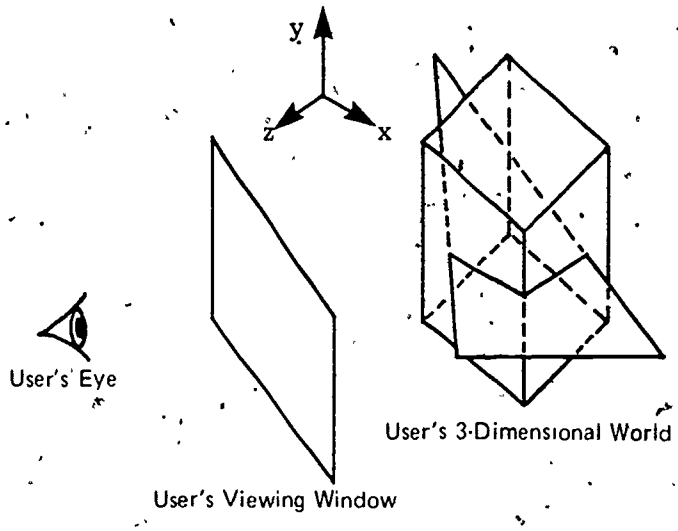


Figure 4 – User in His 3-Dimensional World

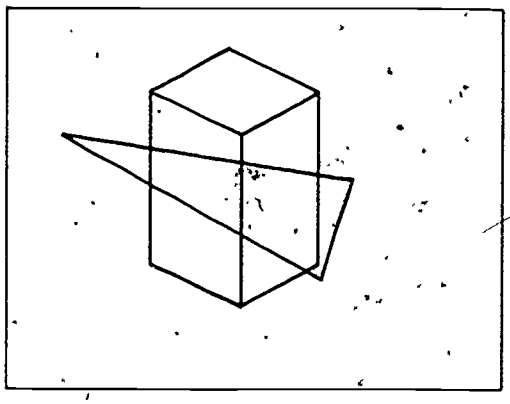


Figure 5 – Projected and Clipped Atoms

decisions have yet been made between the surface of the cube and the surface of the triangular plane.

The visible surface processor resolves conflicts between these overlapping surfaces and encodes the image into the edge form of Figure 6. Note that as conflicts between overlapping surfaces are resolved, intersection edges are automatically created.

Figure 6 also demonstrates the efficient data format at the CHARGE terminal. Each "visible edge" in Figure 6 defines a color and a brightness to its right side; the edges themselves are ordered in x such that for any scanline (y position) all of the edges to the right of Edge i have edge labels greater than i. Edge 1 is a special "marker" edge that flags to the CHARGE terminal decoder unit that there are data present. The image of Figure 6 contains 20 edges requiring that 20.76 = 1520 bits be buffered at the CHARGE terminal

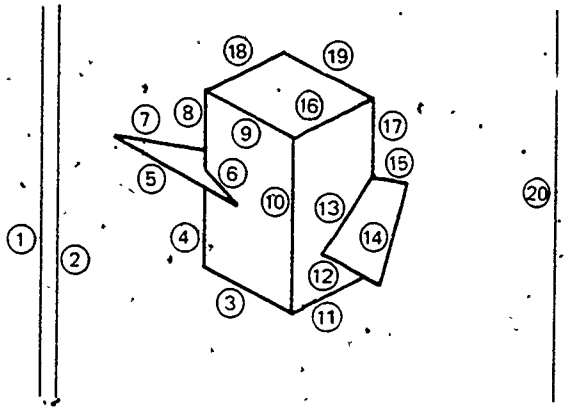


Figure 6 – **Encoded Edge Image**

to produce a sharp, colored video image on a monitor with resolutions 1600 units horizontally by 1200 scanlines vertically (4.3 aspect ratio). If the same image were to be buffered as a video image with 9 bits of brightness and 12 bits of colored information for each point, 20.7×10^7 bits would need to be buffered.

The CHARGE image generator and terminal also have the ability to "smooth-shade" curved surfaces; that is, curved surfaces are approximated by planar facets and the resulting edges have bits set instructing the CHARGE terminal decoder unit to linearly interpolate the brightness between successive edges, thereby giving them the appearance of a continuous, smooth, "round" surface with no edges within it. Thus, round surfaces such as spheres and cylinders appear round on the terminal.

APPROACH

In designing the CHARGE terminal, two problems had to be solved—how to code and buffer a high-resolution colored image, and how to decode and display the image on a CRT, both at a reasonable cost.

The solution to the buffer problem is similar to that used in stroke terminals for line graphic images—encode the image, rather than buffer the video. The solution differs, however, in that there is no practical limit in the number of "picture elements," called edges results, as in the case for a stroke terminal where large numbers of lines cause an objectional flicker. The edge encoding technique permits the display of a 4 10^7-bit image at the cost of a 4 10^5-bit buffer—a key to a low-cost terminal.

The solution to the refreshing (decoding) problem required innovative memory architecture and data organization, as well as innovative decoding hardware, in order to generate video for output on a color monitor without significant constraints in image complexity and resolution.

The following sections describe the terminal's encoding and architecture from a theoretical and detailed logic point of view.

IMAGE ENCODING

EDGE ELEMENTS

An image is encoded by a set of edges. An edge is an imaginary line to the right of which is displayed a color up to the next edge (see Figure 7). In addition to defining the color on its right, an edge defines the brightness along the edge and when paired with the edges on its right defines the brightness in the constant color region between edges.

Both color and brightness parameters are log functions in order to minimize bits required for their storage and in order that brightness and color information can be combined additively rather than multiplicatively. In order to simplify the discussion, the terms brightness and color shall refer to log functions, an explicit reference to linear functions being made when needed.

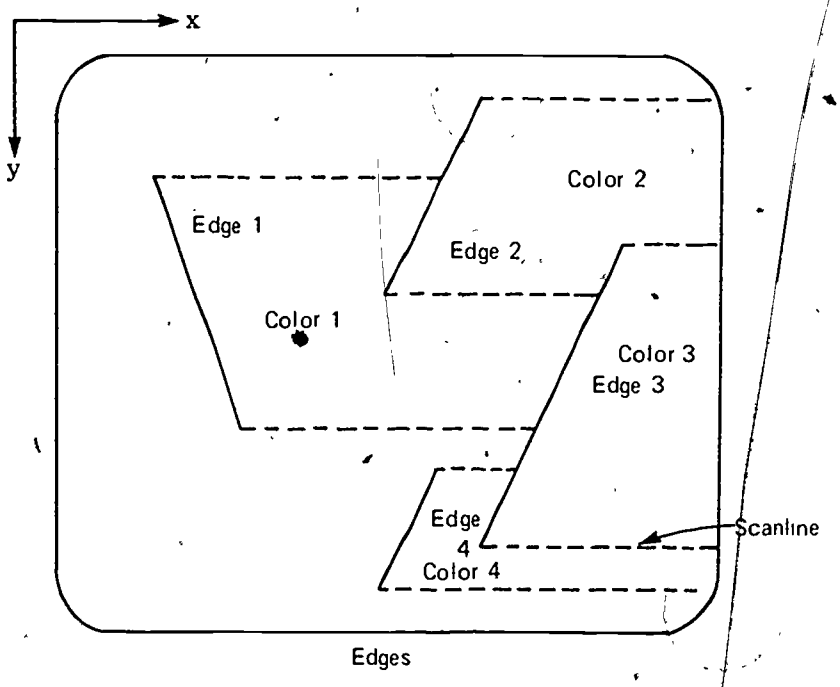


Figure 7 – **Edge Elements**

In Figure 8, where x is positive and increasing toward the right and y is positive and increasing downward, an edge is defined by the 76-bit n-tuple $< X,Y,H,S,B,C,G,F,E >$ where:

(a) $X = x^t$, an 11-bit x-coordinate of the edge's top;

(b) $Y = y^t$, an 11-bit y-coordinate of the edge's top;

(c) $H = y^b - y^t$, an 11-bit height, where y^b is an 11-bit y-coordinate of the edge's bottom;

(d) $S = (x^b - x^t) \cdot 2^{-M+11} / (y^b - y^t)$, a 13-bit signed integer (12-bit integer plus 1-bit sign) representing the slope where x^b is an 11-bit x-coordinate of the edge's bottom and where M is the number of consecutive leading zeros from the left in the 11-bit H up to a limit of 10 zeros;

(e) $B = b^t$, where b^t is the 7-bit brightness (log) at the top of the edge;

(f) C = c, color to the right made up of a 4-bit red, a 4-bit blue, and a 4-bit green component (each log-functions);

(g) $G = (b^b - b^t) \cdot 2^{-M+11} / (y^b - y^t)$, a 9-bit signed integer representing the gradient of brightness defined downward along the edge where b^b is the 7-bit brightness (log) at the bottom of the edge;

(h) F, a 1-bit flag which, if zero, implies that the brightness to the right is to vary linearly (in the log domain) toward the value at the next edge;

(i) E, a 1-bit flag marking the beginning of a set of edges in memory.

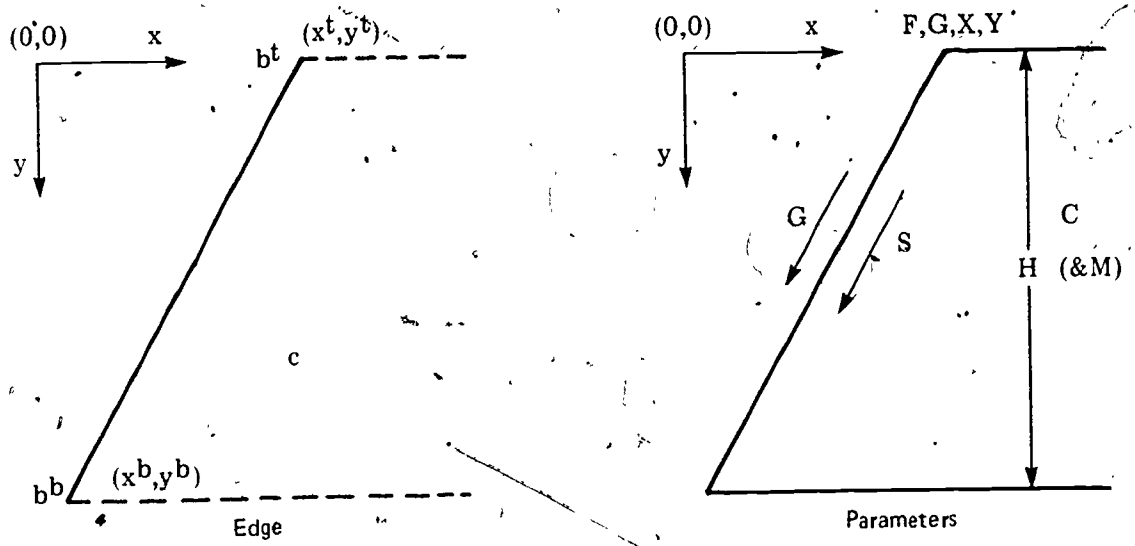


Figure 8 – Definition of an Edge and its Parameters

Portions of the 76 edge words in the terminal are coded in different ways. For instance, Y and H are held in complement form ($\overline{Y}$ and $\overline{H}$). S is held as a magnitude plus sign and where, if S is negative, X must be held in complement form. Similarly, G is held as a magnitude plus sign with B held in complement form if G is negative. For example, a negative S, positive G edge actually is held as the following n-tuple:

$$<\overline{X},\overline{Y},\overline{H},|S| \text{ \& sign, B,G,F,E,C}>$$

The conditional complement is required in order to minimize the hardware implementation costs and has no other effect on the user or system performance. However, in order to keep the discussion simple, the actual form in which data are held will be ignored.

POLYGON SURFACE ENCODING

A polygon of arbitrary shape and color can be represented by edges. In Figure 9, a perspective of a cube is represented by 10 edges (one edge is used to define the background).

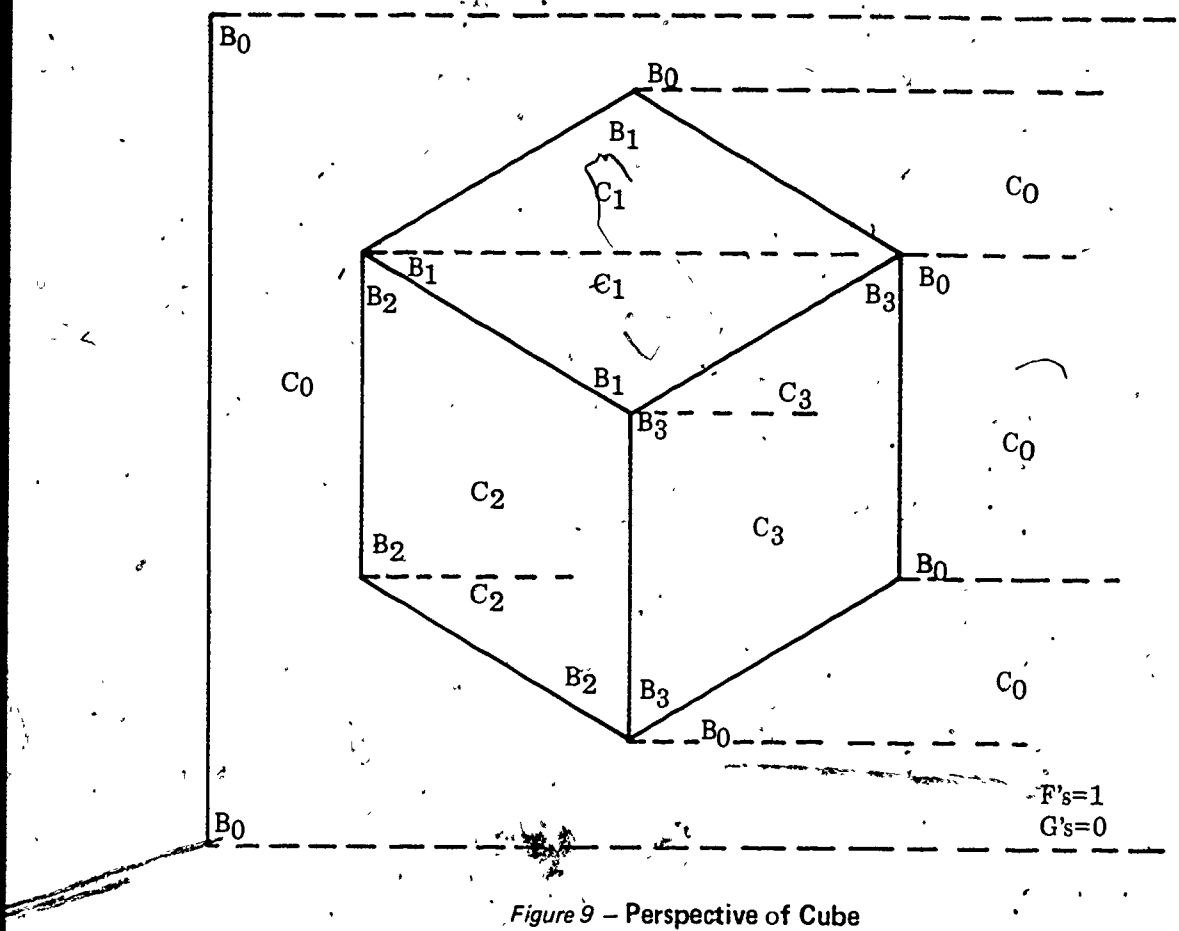


Figure 9 – Perspective of Cube

Because the brightness may vary linearly along edges bounding the polygon and linearly in x between edges, a polygon can represent a portion of a curved surface, as shown in Figure 10.

By means of these polygons, both flat and curved surfaces can be represented in any illumination environment, limited only by the edge buffer size and the number of bits used to define an edge.

FLAT SHADED SURFACES

For a planar surface and point light sources at infinity, the edges bounding the left side of a polygon require F to be 1 and G to be 0. The result is a polygon of arbitrary but constant brightness and color, as seen in Figure 9. The F, B, and G of the edges bounding the right side of this type of polygon do not influence the brightness within the polygon.

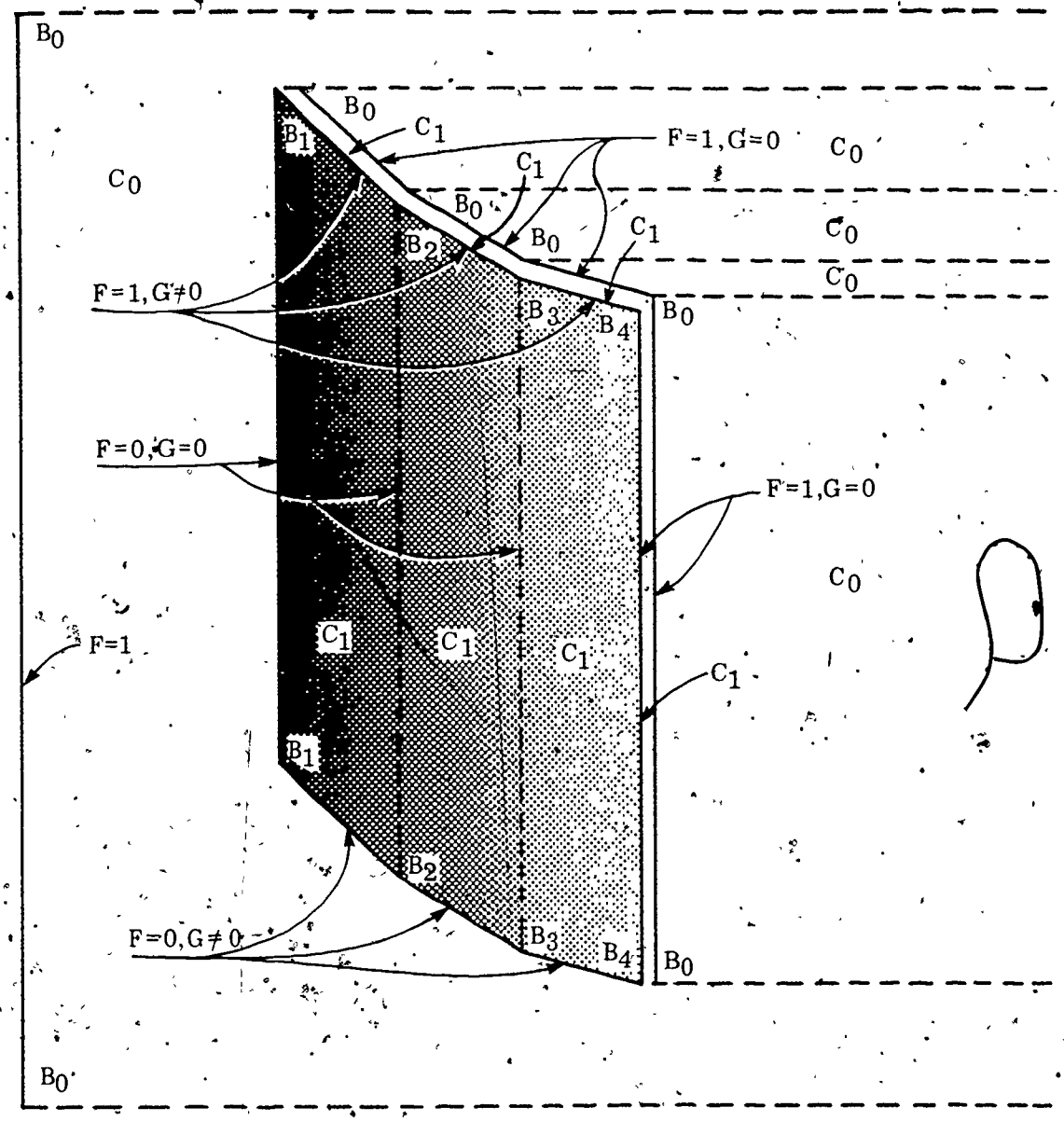


Figure 10 – Perspective of Curved Surface

A planar surface of an object has a constant brightness only for light sources at infinity. Planar surfaces that are illuminated by neighboring light sources require a varying shading and thus fall into the category of curved-shaded surfaces.

CURVED SHADED SURFACE ENCODING

For planar surfaces illuminated by neighboring light sources and for curved surfaces, the brightness throughout a polygon must vary. The brightness within the polygon is a

piecewise linear function of that at the corners of the polygon because of the way brightness is defined along and between edges. In particular, the brightness along an edge at scanline y is defined by:

$$b(y) = (y - Y) \cdot G \cdot 2^{M-11} + B \qquad \text{(Equation 1)}$$

which reduces to

$$b(y) = (y - Y)(b^b - b^t) / (y^b - y^t) + B \qquad \text{(Equation 2)}$$

which is the linear interpolated brightness along the edge.

If on the same scanline the brightness of two adjacent edges are $b_1(y)$ and $b_2(y)$, then the brightness at x between the two adjacent edges bounding a curved shaded surface is defined by

$$b(x,y) = (x - x_1)(b_2(y) - b_1(y)) / (x_2 - x_1) + b_1(y) \qquad \text{(Equation 3)}$$

where x_1 and x_2 are the x intercepts of the two edges and where the form of the equation is that of a linear interpolation, as shown in Figure 11.

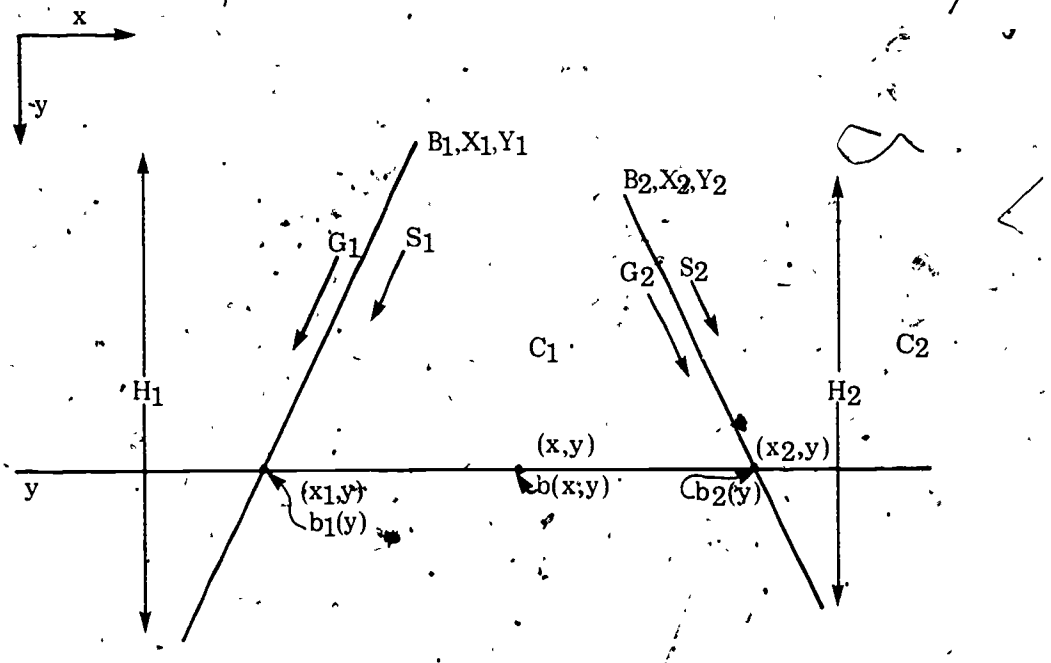


Figure 11 – **Curved Shaded Interpolation**

The edges on the left of curved shaded polygons have their F set to 0 (see Figure 10). The edges on the right of these polygons must, of course have the appropriate Gs and Bs. However, their Fs depend upon the nature of the surface to the right. If a discontinuity in brightness is required to the right of a curved shaded surface, the edges on the right of

the shaded polygon must be followed by another set of parallel edges which redefine the brightness and color farther to the right (see Figure 10).

ENCODING COMPATIBILITY TO IMAGE GENERATION

Aside from the obvious cost advantage afforded by the data compression of edge encoding, there is a second advantage related to image generation.

The generation of 2-D perspectives out of 3-D objects requires planar approximations to curved surfaces in order that low-cost, high-speed, visible surface calculations can be achieved. The perspectives of these planar surfaces are polygons that are conveniently represented by edges. Also, the brightness of the 3-D objects is generated only for the corners of the facets representing curved 3-D surfaces where linear interpolations in the perspective domain are assumed for all other portions of the surfaces. This interpolation is identical to that used at the terminal, permitting a perfect match between image generation and terminal decoding.

VIDEO TO EDGE ENCODING

In theory, it is possible to represent any video image to any accuracy by edge encoding. However, at the present time, no algorithms have been developed.

Because of the lower data compression often afforded by storage of one or more 2-D perspectives in edge format than afforded by the storage of 3-D objects (with predefined descriptions) and because of the inquiry flexibility afforded by generation of 2-D perspectives from the 3-D description, the need for video-to-edge algorithms is infrequent.

TERMINAL ARCHITECTURE

The terminal has four major components—main memory, memory decoder, display monitor, and communication controller (see Figure 12).

The memory buffers the edges, and for each scanline outputs to the decoder only those that are valid (i.e., cross the scanline).

The decoder for each valid edge derives the x and b intercepts, forms segment data (portions of a scanline between edges), and after temporarily buffering that data for one scanline, generates the three-color components for output to the display monitor.

The display monitor differs from a commercial color TV receiver in that the bandwidth of the video amplifiers is increased by a factor of 5, and anti-log amplifiers are

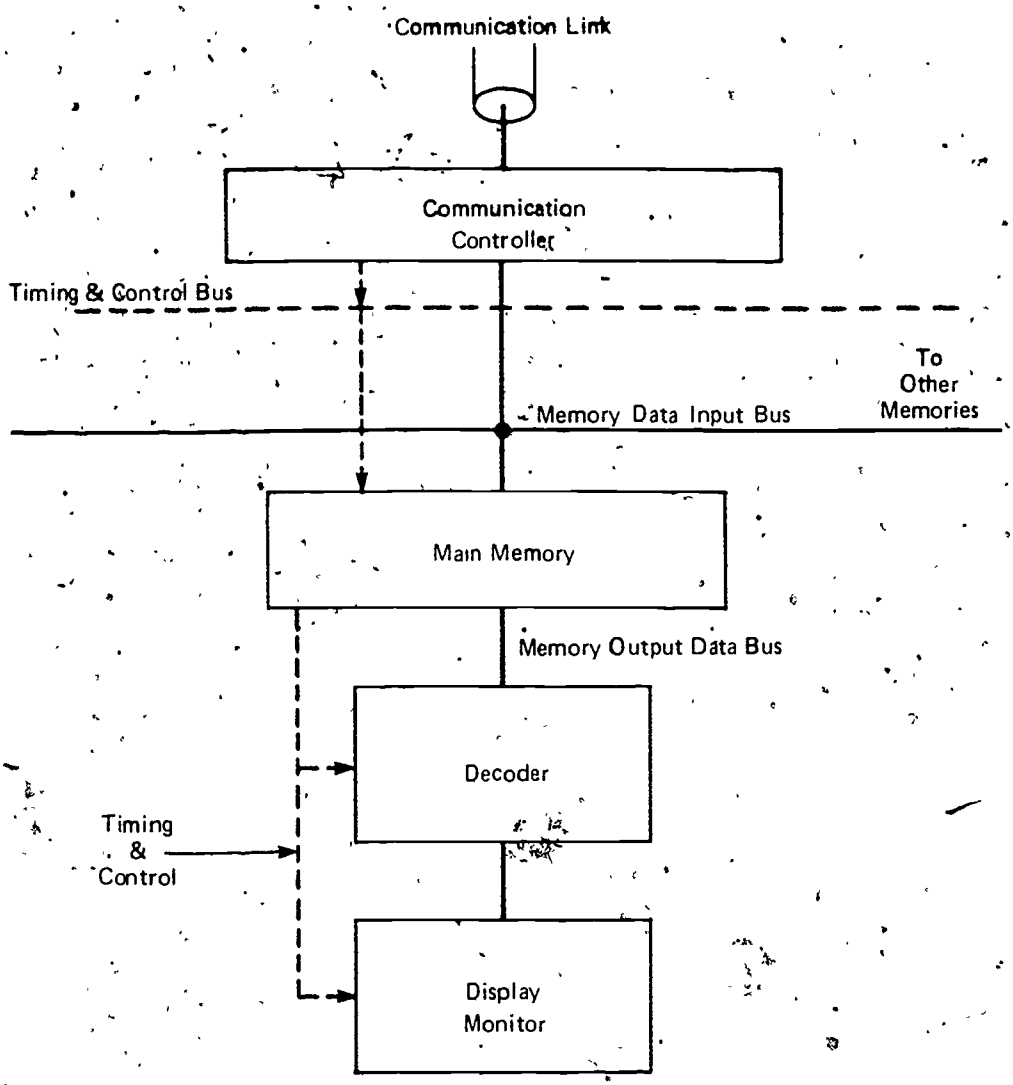


Figure 12 – **Terminal Block Diagram**

added ahead of the video amplifiers. Vertical "resolution" is increased to 1200 by means of a 5:1 interlace which does not alter the horizontal and vertical scan frequencies from that of commercial TV.

The communication controller interfaces one or more memories to a coax link and supplies basic timing and control signals for one or more terminals.

Each of the major components is described in detail below.

MAIN MEMORY

In order for the memory to deliver to the decoder the edges crossing a scanline, a large block of edges must be able to be scanned during the scanline period of about 64 microseconds. In order not to have to scan all of memory during the scanline period, the memory is organized into two sections, one that is scanned (read) once for each scanline period and one that is scanned once for each vertical trace period (every 1/60 second), as illustrated in Figure 13.

MOS dynamic shift registers are employed, not only because of their low cost but also because of their high speed.

As shown in Figure 13, memory is organized into N banks, each bank organized into the two sections referred to above. A unibus output structure is employed from main memory to the decoder for convenient memory expansion and simple interfacing. Similarly, an input bus is employed to all N banks of memory. In addition to the N banks, a memory/decoder control unit is required.

Figure 14 is a block diagram of a bank of memory. The section of memory that is scanned (cycled) in each scanline period consists of 76 256-bit shift registers, and two 76-bit Transistor-Transistor Logic (TTL) registers. These shift registers and TTL registers form a 258 cyclic edge memory. The larger section of memory, which is scanned once every vertical trace period, consists of 76 1024-bit dynamic shift registers. It can be functionally inserted into the 258-bit memory loop, forming a 1282 cyclic edge memory during certain scanline periods.

In addition to the two sections of memory, there are (a) a scanline cross-detection circuit with a control circuit and a delayed output register, and (b) a bus interface. Because the larger section of memory, blocks of edges can be moved in and out of the smaller section of memory during certain scanline periods without altering the order of the edges.

The 258 edge memory is cycled each scanline. This 258-edge memory loop is always clocked (cycled) completely around between the insertions of the larger section of memory to keep the data in the same order.

Because the N banks of memory are clocked together during image decoding, blocks of up to N . 256 edges can be moved into the smaller sections of memory during a scanline period.

A portion of the edge data passing through the output register of a bank is fed into the scanline cross detection circuit, which compares the y of the scanline with both Y

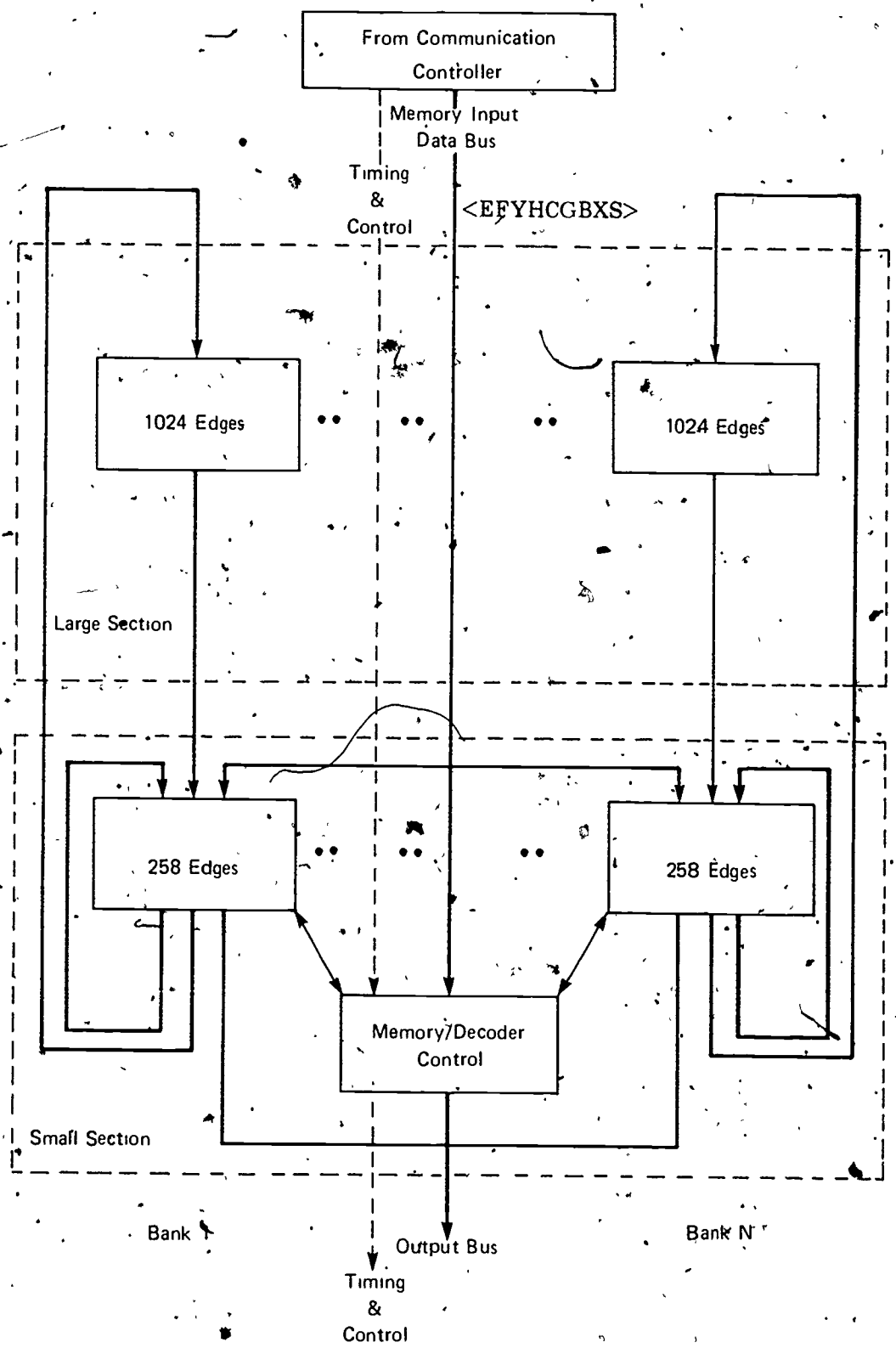


Figure 13 – Main Memory

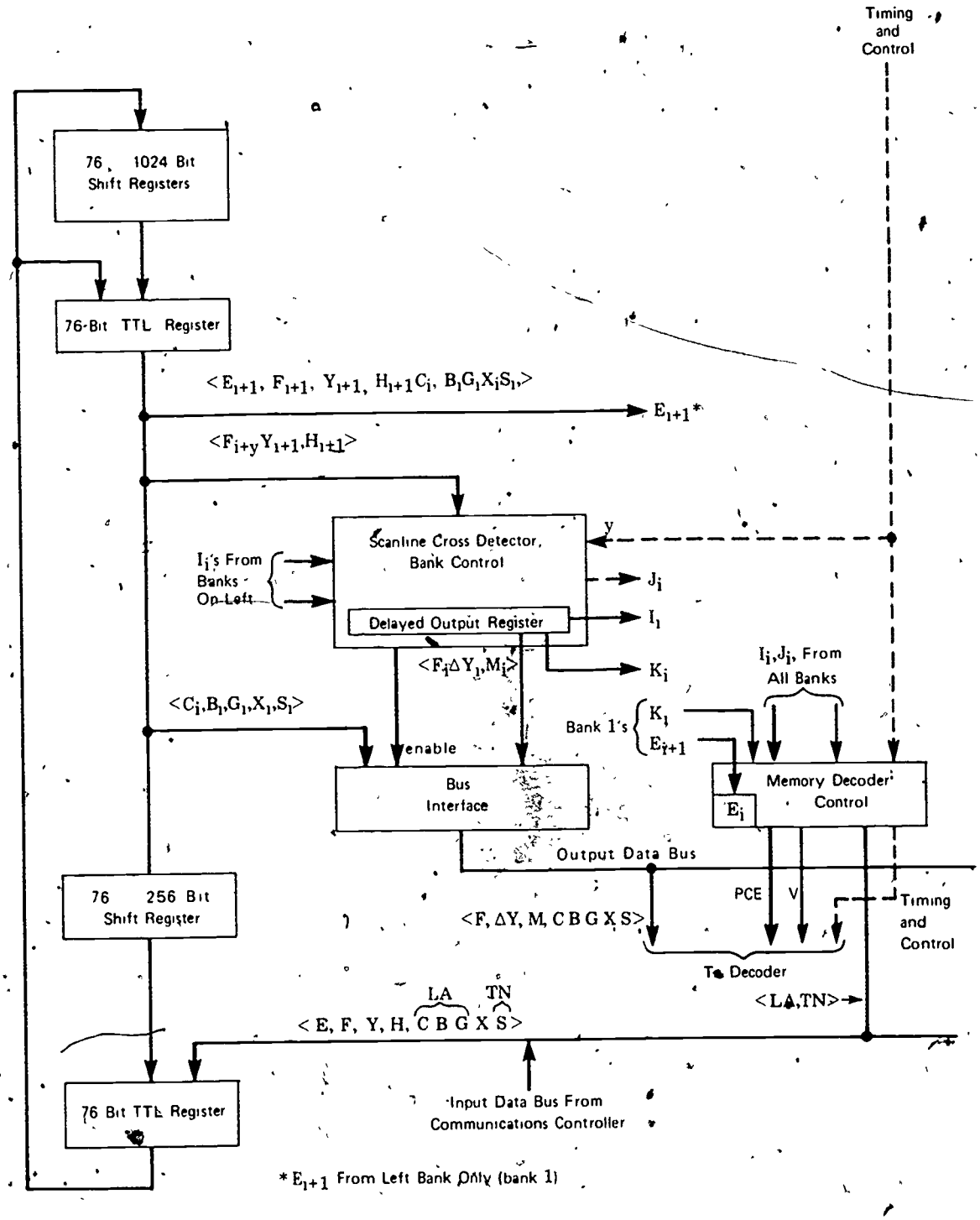


Figure 14 – **Block Diagram of Nth Bank of Memory**

28

and Y + H of the edge. Results of this comparison are loaded into the "delayed output register" of the scanline cross circuit exactly one memory clock time later when new edge data enter the output register. The contents of the delayed output register at this time are $\Delta y (= y - Y)$, M (the number of consecutive left most 0s in H up to 10 zeros), F, K, and I. I is a 1-bit flag which, if set, means that the edge is crossing the decoded scanline, as shown in Figure 14. K is a 1-bit flag which, if set, means this is the last scanline that this edge will cross.

In order that the ΔY, M, F, K, and I data for an edge be available in the delayed output register at the same time the S, X, B, G, and C data for that same edge are in the output register, a word of memory contains portions of data for two edges, the Y_{i+1}, H_{i+1}, and E_{i+1} data for an edge i+1, and the S_i, X_i, B_i, G_i, and C_i data for the preceding edge i. This eliminates the need for a delayed output register which must hold the data for an entire edge. This staggering of edge data is described in more detail in the next section of the report. E_{i+1}, rather than being buffered in the delayed output register, goes directly to the memory/terminal control unit where it is buffered. F_i is buffered along with ΔY_i and M_i in the delayed output register because one spare register bit is available there with an interface to the bus. (11-bit Δy and 4-bit M leaves 1-bit when 4-bit register chips are used.)

The delayed output register has tri-state outputs for the direct interfacing of F, ΔY, and M to the memory output bus. The remaining portion of the edge data residing in the output register is interfaced to this same bus through tri-state logic bus interface hardware.

The edge data are enabled onto the bus under the control of the I flags of the N banks. Because more than one edge may be valid among the N edges of N banks, the banks are each given a different priority corresponding to the x ordering of the edges in the memory to be described in a later section. Thus, if more than one edge is valid, one by one they will be enabled onto the memory output bus in the proper x ordered sequence.

The control circuit in each memory bank receives the I flags from the banks on its left. When a memory bank's I is 1, and those of the banks to its left are 0, it places its data on the bus and resets its I flag, thereby permitting the other banks to gain control of the memory bus in turn (see Figure 14).

In addition to the control circuit in each bank, the memory banks are under control of a memory/decoder control unit that clocks and controls the components of the N

banks and the decoder. This control unit receives I and J flags from each bank, and if none of the N edges that are clocked into the output registers are valid (all Is = 0), flags the bus data as invalid. A J flag equal to 0 means that I = 1 and that there is an I to the left that is 1. Although this information can be rederived from the Is by the memory/decoder controller, it happens to be already available in the bank controllers. A J = 0 means that there is yet another valid edge to be enabled on the bus and, thus, that memory must not yet be clocked.

The E and K bits from the left bank are used by memory control to locate the first, and thus the last, edges within subsets of edges. Before describing further the operation of the memory, the organization of the data within memory must be described.

Data Organization

Edge Blocking. The data in the two sections of main memory must be blocked in y and ordered in x so that up to 256 valid edges may be detected and fed in x order to the decoder every scanline period.

The original set of edges cannot be partitioned into disjointed groups, where each group can be used alone to decode a range of y of the image and where all portions of the image are represented by the groups. The process of blocking the edges transforms this original set of edges into a larger set containing duplicates of some of the original edges where the edges of the larger set can be partitioned into the required type of disjoint groups. This partitioning or blocking of edges (a) puts the edge data in a form where only one portion (block) needs to be decoded at a time and (b) orders the blocks so that blocks of edges can be exchanged between the large and small sections of memory in such a way that the smaller section of memory always holds the block of edges that was used to decode the previous scanline and that may be needed to decode the next scanline.

The edges within a block may extend above or below the range of y over which the block is used for decoding.

Each block of edges must include at the beginning a marker edge and at its end a small number of null edges. The marker edge is defined to cross those scanlines over which a block is defined as valid. When this edge is detected during a scanline decode period as crossing a decoded scanline, it signals that the following edges are valid for decoding. If the marker edge is detected as valid for the last time, ($K_i = E_i = 1$), the memory controller sets a flipflop so that a new block of edges will be used during the next scanline decode period.

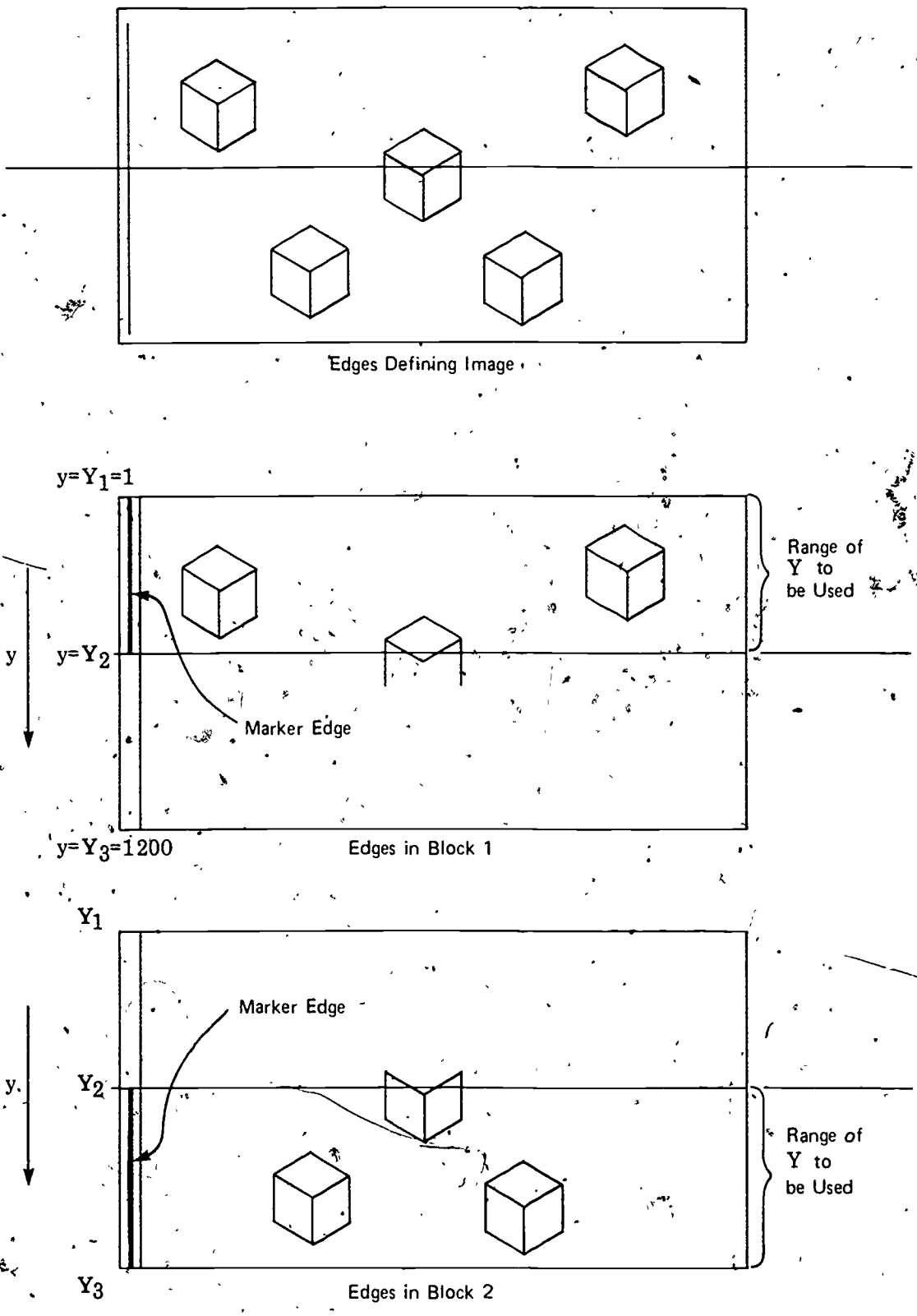


Figure 15 – **Example of Blocking**

Null edges are used to occupy unused memory locations. A null edge is defined as one whose Y is impossible (i.e., $1313 < Y < 2^{11}$).

Blocks are formed out of the original set of edges as follows: First, the edges are y ordered from top to bottom of the screen using their Ys, blocks are formed from the top of this list and from those edges used to fill the preceding block, which extends below the range in y over which that block is valid for decoding.

Starting with block 1, whose range begins at $y = Y_1 = 1$, edges are moved into this block from the y ordered list where all edges starting on the same scanline must be moved together as a group until the number moved into the block exceeds the function f where:

$$f = (257 - (Y_2 - Y_1) / 80 - 1) N - 1 \qquad \text{(Equation 4)}$$

where Y_2 is the Y of the last group of edges moved. This last group of edges that overflowed the block is put back in the y ordered list and a marker edge is added with $Y = Y_1$ and $H = Y_2 - Y_1 - 1$. Then null edges are added to fill the block to an integer multiple of N. All edges in this block that extend to or beyond Y_2 are copied into block 2.

Block 2, now containing those edges copied in from block 1, is then filled from the y ordered list until a group of edges causes it to overflow. Just as in block 1, the last group is put back in the y ordered list, a marker edge and nulls are added, edges extending too far are copied out, and so forth, until memory is filled. Figure 15 illustrates an image broken into two blocks where five edges are common to the blocks. It will be shown later that a minimum of six blocks are necessary to fill up memory.

Out of the original set of edges sufficient to define the image, up to four times 256 edges may be duplicated in blocks 2 through 5, where four smaller sections of memory fit into the larger section of memory, and where 256 is the maximum number of edges allowed to cross a given scanline.

Thus, blocking edge data forces the terminal to have a slightly larger buffer than the theoretical minimum. Were memory not organized into two sections but rather into only the smaller section and were N made five times larger, memory would consist of only one block, eliminating the need to duplicate some of the edges. The trade-off between the greater overhead costs per bit of a large N uniblock memory and the greater cost of more bits of a small N multiblocked memory favors the blocked memory. This is because about half as many chips are required, because memory will be extendable by the user of longer shift registers as they become available without significant chip count

increases, and because the output stages of the image generator can work with smaller chunks of edges.[1]

Block Organization and x Ordering

After memory is loaded, it appears as in Figure 16. The bottom layer of memory is the set of N input registers, the next 256 layers are in the N small sections of shift register memory, the next layer is the set of N output registers, and the remaining 1024 layers are in the N large sections of shift register memory.

Data paths are indicated by the arrows on the left side and the arrows down the middle of the memory sections. When the output registers select the data from the input registers, a 258 layer memory loop is formed holding N . 258 edge words. This is the mode for reading the smaller section of memory. When the output registers select the data from the 1024 layer section of memory, a 1282 layer memory loop is formed, holding N . 1282 edge words. This is the mode where the large section of memory is being read. Data outputting from the memory come from the output registers and data inputting to the memory enter the input registers, replacing the data that would have come from the 256-layer section of memory.

During loading, data enter at bottom and follow the data path of the 1282 memory loop (i.e., to the top, then shifting downward).

Notice the staggering of the edge data where the data for an edge reside in two words of memory. During loading, edge data are not staggered. Upon completion of loading, the left portions of all N banks are clocked once more, thereby producing the desired staggering.

The memory is cyclic so that there need not be any unused memory locations. Also, the image generator can unload its blocks of data, each as they are formed without the need for buffers which hold the entire image.

Edges in each block are x ordered from left to right and upward as indicated in block 1, where there are L_1 edges and where L_1 / N is an integer. The first word of a block must be in the left bank of memory and the last word of a block must be in the right bank of memory, the first word is flagged as such by a 76th bit, E, as shown in blocks 1, 2, and K.

The definition of the "partial" x-ordering is that edge j must follow edge i if edge j crosses the same scanline as edge i and edge j's x intercept is greater than that of

[1]The number of blocks is not limited to five, (actually six). Blocks containing as few as N edges are allowed.

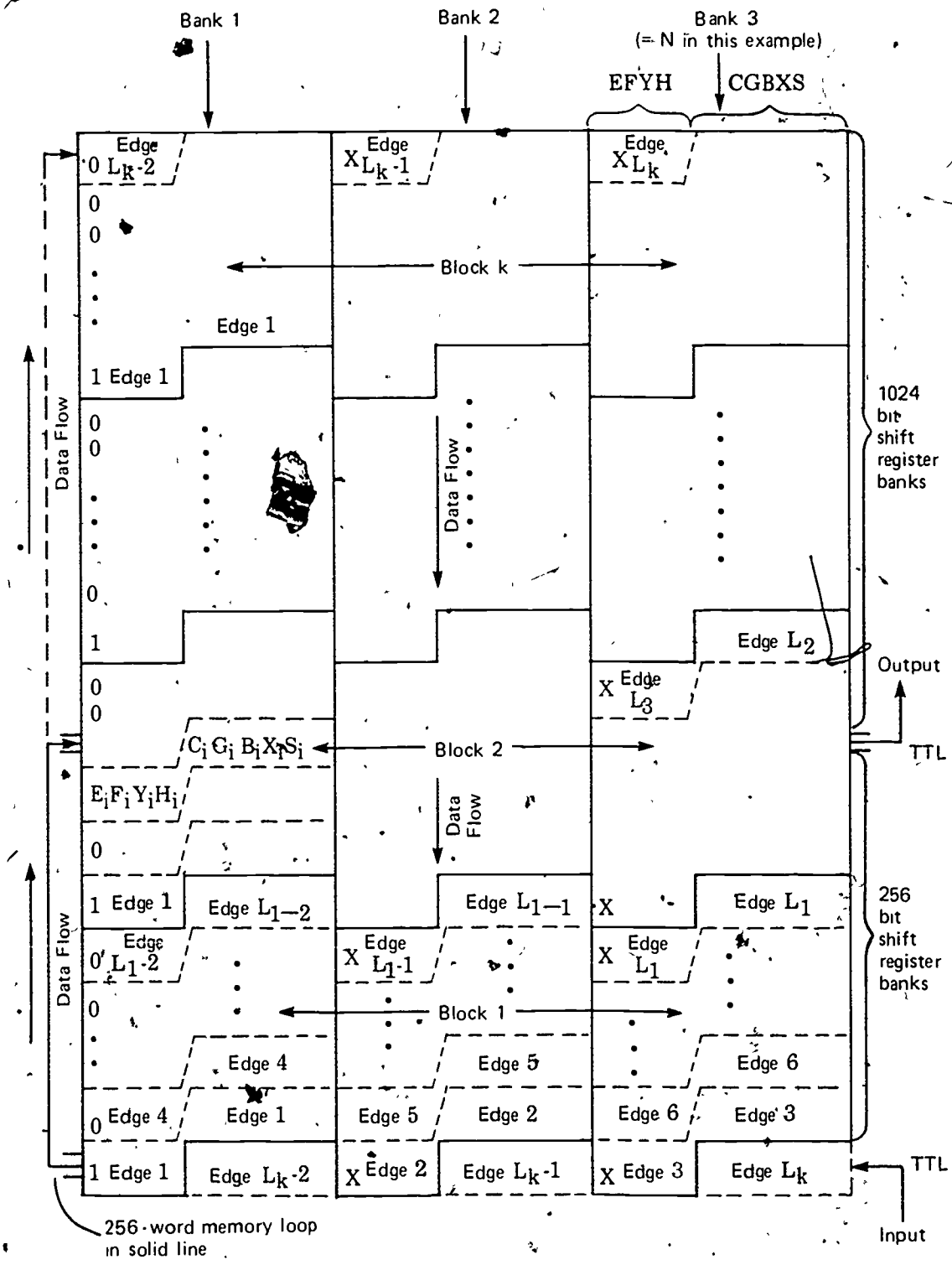


Figure 16 – After Loading, or When Sections Are Aligned

edge i. (Refer to Figure 7 where edge 3 must follow edges 1, 2, and 4, where edge 2 must follow edge 1, but where edge 4 need not follow edges 1 and 2).

As indicated by the data flow, there are two sources of input to the output register and two sources of input to the input register. Normally, the output register is selecting input from the input register forming a 258 . N word loop where the smaller section of memory will be read whenever memory and the two TTL registers are clocked. On approximately 16 scanlines, the output register during a portion of scanline decode period selects input from the larger section of memory forming a 1282 . N word loop emphasized by the dashed arrows in Figures 16 through 19. These occasions when a

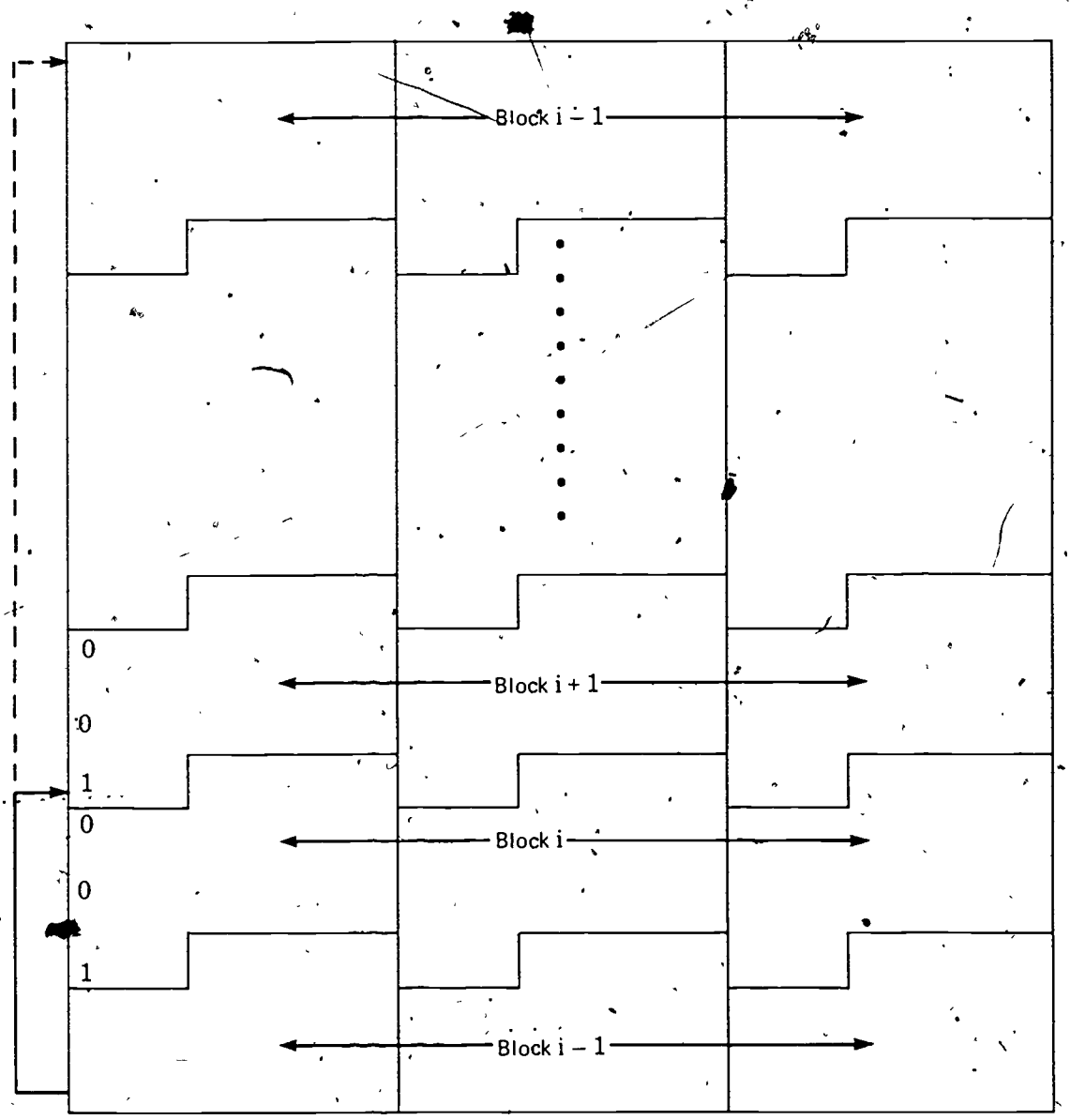


Figure 17 – At End of First Scanline Decoding Block i

1282 . N word loop is selected and the larger section of memory is clocked along with the smaller section of memory (i.e., when the larger section of memory is being read) are either when an entire block of edges is being exchanged between the two sections of memory or when N edges (one layer of words of a block) must be exchanged in order to "revitalize" the dynamic memory of the larger section of memory. The latter single layer exchanges move data from the next block located in the larger section of memory into the smaller section of memory ahead of the time it is needed for decoding. Thus, the

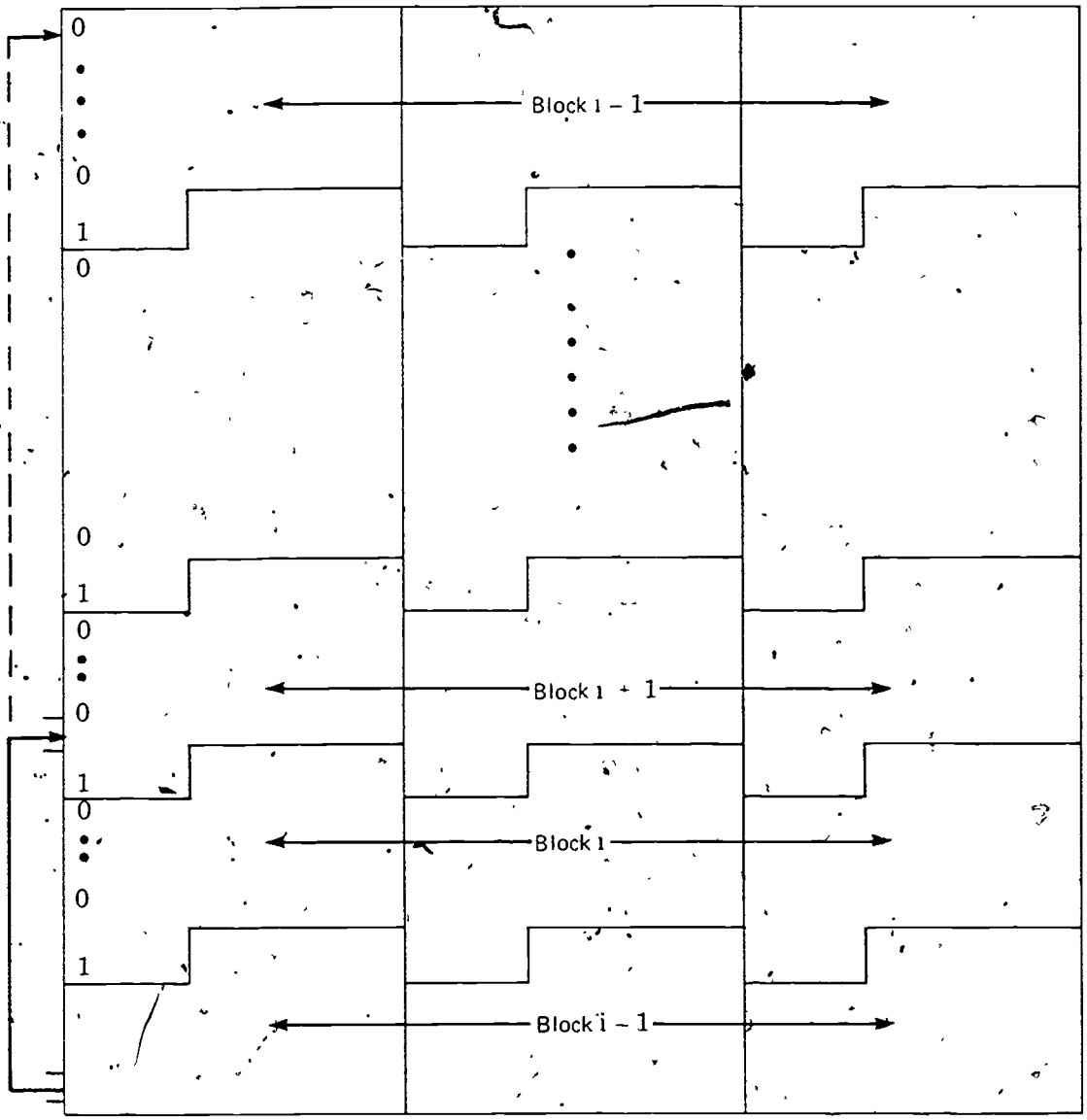


Figure 18 – Early Portion of Scanline Decoding Cycle

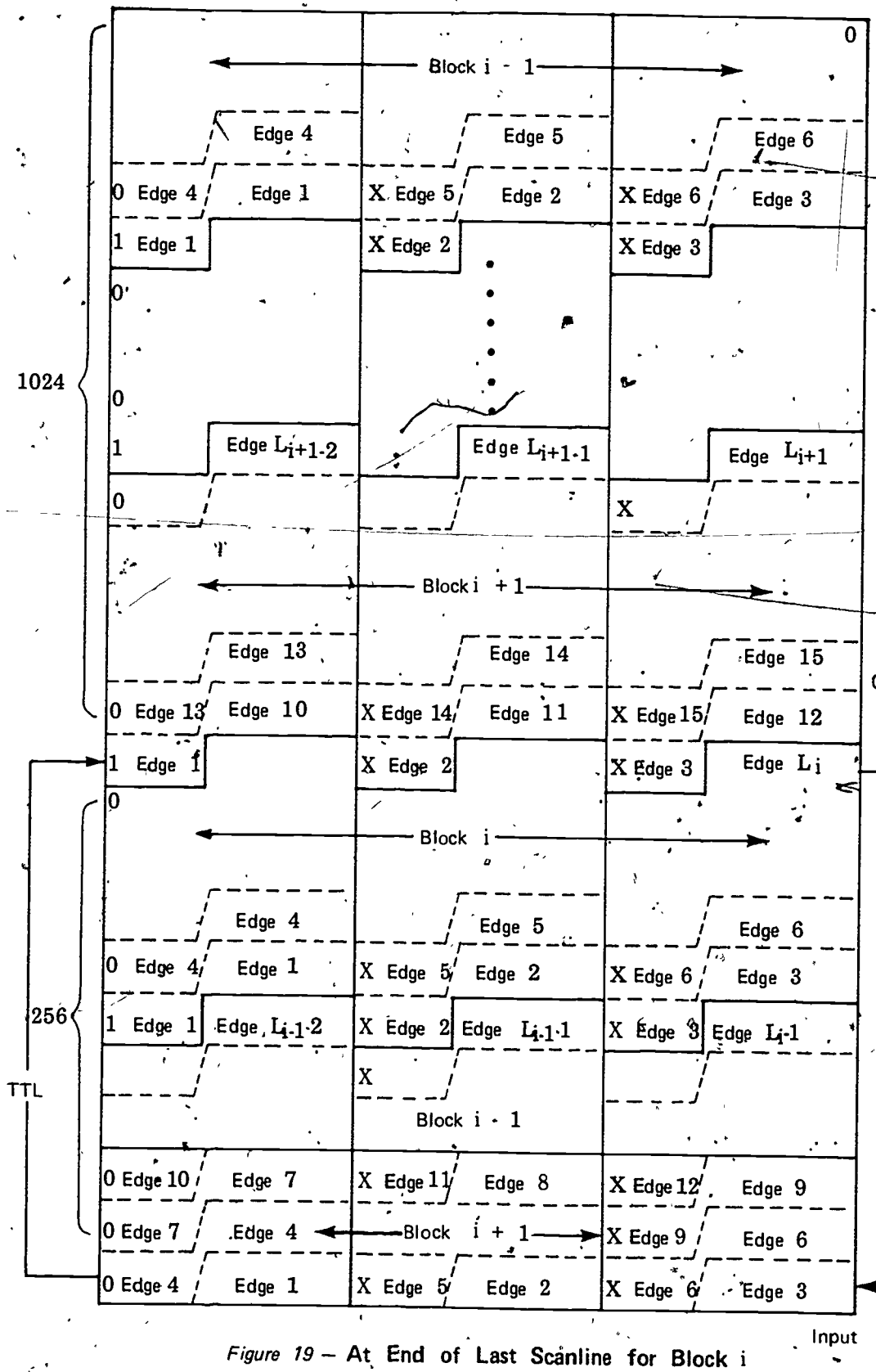


Figure 19 – At End of Last Scanline for Block i

block of edges cycling in the smaller section of memory must be small enough to not be shifted out by these early entries. Thus, function f in Equation 4 has included a term to guarantee the integrity of a block of data in the smaller section.

After memory is loaded, a memory loop flipflop has been set forcing the memory into a 1282-layer loop so that data passing through the output register during the next scanline period is not from block 1, and likely not from all of the first block encountered (block 2 in Figure 16), because part of it will already have been moved into the smaller section of memory. This is the *only* scanline decoding period where an entire block of memory does not pass through the output register, and should really be considered a part of the loading cycle.

At the end of the first scanline period following loading, and at the end of all other scanline decode periods, memory always appears as in Figures 17 or 19, where Figure 19 differs from Figure 17 because of the "early reading" of portions of the next block required to rejuvenate the dynamic memory. The last edge of the block of edges just decoded is in the output registers at this time. The E flag of the first edge of the next block is used to end the cycling of the memory during a scanline decode period. If invalid data happen to be in the memory, such that there is no flag set to mark the beginning of blocks, then the memory cycle is terminated after 260 memory clocks. This protects the smaller memory from excessive average clocking frequencies, a situation that would destroy that memory.

The data from block i + 1 are always put behind the block i data in the smaller section of memory and that the block i data have entered behind that of block i - 1, most of which have been pushed out and around into the larger section of memory.

Because the end of the data for block i + 1 which has entered prematurely is not marked by a flag, a 258 state alignment counter is held to a value of -1 whenever the larger loop is clocked. Otherwise, this counter always increments along with the clocking of the smaller section of memory (cycling through values -258, -257, . . . -1) so that a count of -1 flags that the data between the two sections are "realigned," that is, the end of block i + 1's data in the smaller section of memory have entered the output register. Only when this counter is -1 is the larger section of memory allowed to be selected and clocked.

If during a scanline decode period where block i is being decoded the first edge (marker edge) is detected as valid for the last time (K = 1), the memory loop flipflop is set, otherwise it is reset. Thus, if at the beginning of a scan decode period the memory loop flipflop has been set or if 16 scanline decode periods have passed without reading

the larger section of memory, the larger loop will be read when the alignment counter has reached -1 either at the beginning of the scanline decode period as in Figure 17 or further along as in Figure 18, where at the beginning of the scanline period memory appeared as in Figure 19. That is, during a decode cycle requiring the decoding of block i + 1, most of which still resides in the larger section of memory, the small section of memory is read and decoded until the counter is -1, after which all further reading is from the larger memory. In other words, at the beginning of a cycle, the image in memory appears as in Figure 19, and when the image appears as in Figure 18 where the counter is -1, the larger memory loop is read until the image appears as in Figure 17 where i + 1 is substituted for i. Similarly, during a decode cycle requiring a single revitalizing read of the large section of memory, the larger loop will be read for one memory cycle when the counter reaches -1.

After the first memory cycle clocking, the contents of the output register contain the X, G, S, B, and C data for the first set of N edges to be outputted where the Y, H, F of these edges has passed through the cross-detection circuitry producing ΔY, M, F, and I data in the delayed output registers. Normally, these first N edges belong to either block i + 1 or block i - 1, block i always residing near the top of the small section of memory at this time. Thus, if this is a cycle where block i is to be decoded again, sets of N edges are scanned by at peak rate, the memory controller flagging them as invalid to the memory bus, until a "first valid marker edge" in block i is detected. Only the marker edge of block i will be detected as valid.

When the flag is detected, normal crossing detection control of memory clocking is activated so that each of the valid edges (those crossing the scanline) are enabled onto the bus before memory is clocked again (using Js). When a marker edge is detected (valid or invalid) following a valid marker edge, or when 260 memory clocks have occurred, the cycle ends. During scanline decode periods when a new block of edges is sought, the cycle ends on the first marker edge found whether or not a valid marker edge has yet been encountered. The extra "overflow" counter which signals the end if no marker edge is detected guarantees that the 256 shift register system does not clock in excess of an average of 4 megacycles.

There should always be a valid scanline crossing of a marker edge during a decoding cycle if the data have been properly formatted. If not, the memory loop flipflop is set, forcing a new block to move down during the next scanline decode period. This also permits the memory to quickly find the correct block after a loading cycle, no

more than K (number of blocks) scanline periods being required to hit the right block out to K blocks. However, the large section of memory is not allowed to be cycled more than 64 times during a vertical trace period. This protects the larger shift register system from clocking at higher rates than their clock drivers can stand. Again, this protects memory in cases of invalid or incorrect data residing in the larger section of memory where each block moved down may not have any valid edges.

Further Discussions of Memory Operation

Clocking of the memory occurs at multiples of a 134-nanosecond period (134 nanoseconds determined by the maximum frequency of the shift registers). During each period of 134 nanoseconds, one of the N edges read into the output registers will be put on the bus and flagged by the 1-bit V as valid or invalid. If among the N edges, P are valid, then P x 134 nanoseconds will pass before memory is clocked again. If among the N edges none are valid, the edge from the right bank is placed on the bus and flagged as invalid.

During each scanline decode period the smaller section of memory is clocked 258 times and the larger section of memory is clocked up to 257 times once if for refresh purposes and up to 257 times if a new block of edges must be read.

After 258 memory clocks, up to 256 valid edges and up to 258-256 / N invalid edges may have been placed upon the memory output bus. Thus, if all N valid edges happen to be together in only 256 /N layers of memory, a minimum of 514-256 / N clock periods of 134 nanoseconds each is necessary to decode this worst case number and distribution of valid edges within memory.

After each scanline decode period, the small section of memory contains the last block of edges decoded. If a marker edge was detected as valid but not for the last time during this last decode period, only the small section of memory is read during the next scanline decode period, the larger section of memory being read possibly once if it has not been read for 16 scanline decode periods.

If after a scanline decode period a marker edge was detected as valid for the last time, the smaller section of memory is read until the data in the small section are aligned with those in the larger section of memory, after which the larger section of memory is read, the larger section of memory clocking along with the smaller section until the small section has been clocked possibly up to 258 or even 260 times, depending upon the condition that ends the cycle. There are only two possible reasons why no edge crossings will be detected during a scanline decode period. Either the data in memory are

invalid (because of unknown memory state just after turn-on, because of a transmission error, because of a data format error), or the data are valid but the edges are not (no crossings detected) because memory has not caught up to the beam y value.

TERMINAL MEMORY DECODER

The decoder consists of three main components, an edge-to-segment decoder, a segment buffer, and a segment-to-video decoder (see Figure 20). A segment is defined as

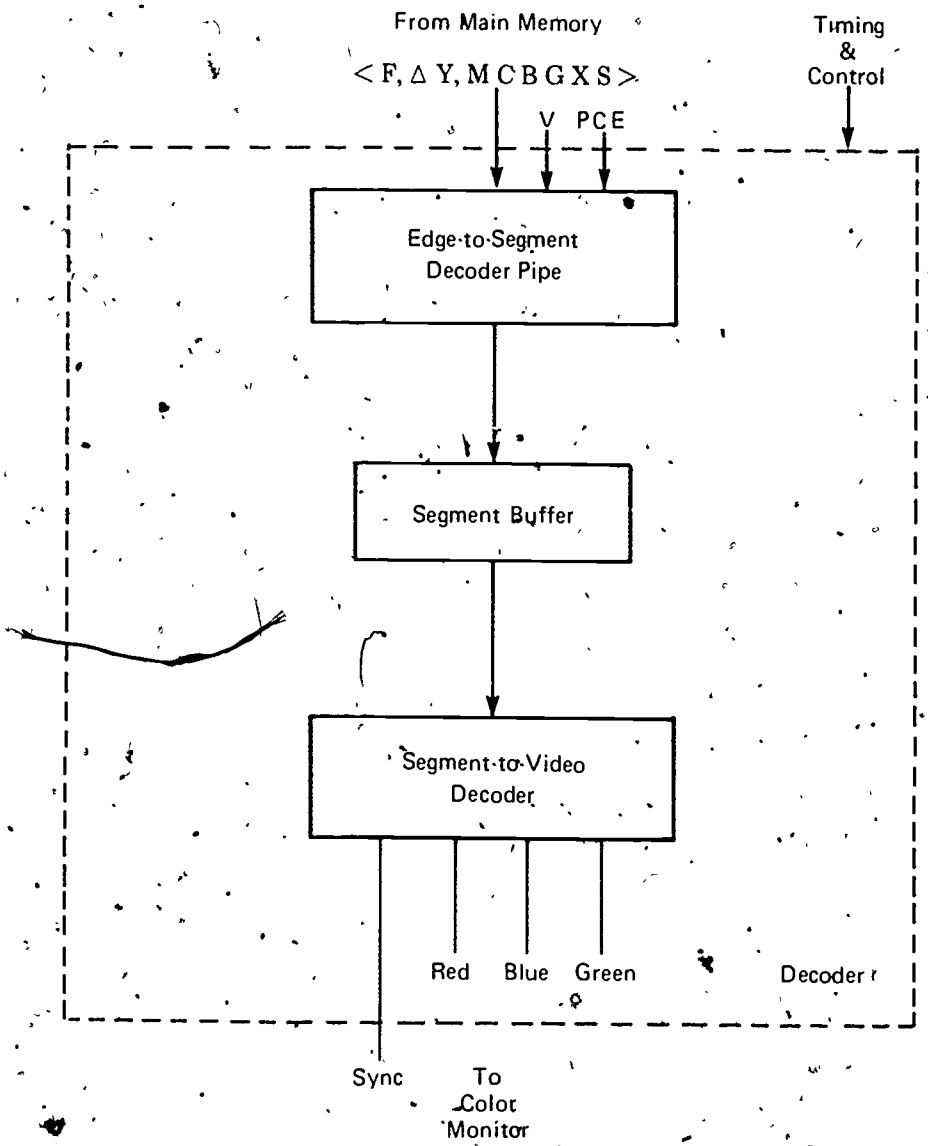
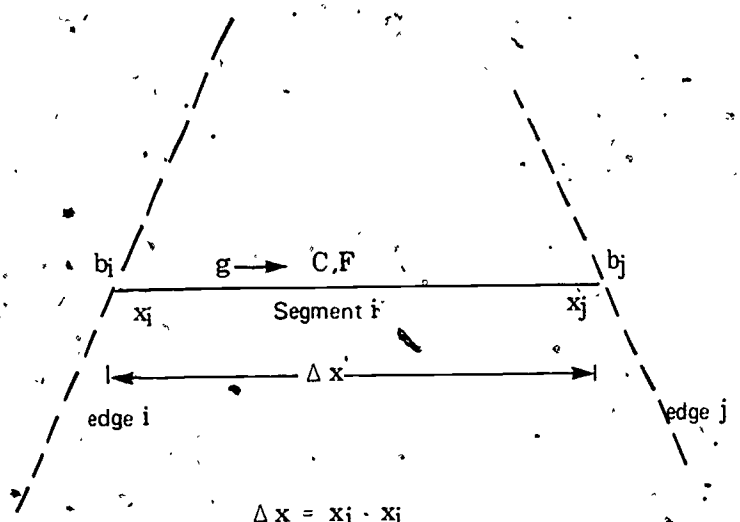


Figure 20 – Terminal Decoder

the scanline interval between two adjacent edges. The parameters used to define a segment are its width Δx, its color C, its normalized gradient of brightness g (10 bits), and the smooth shading bit F (see Figure 21).



$$\Delta x = x_j - x_i$$

$$g = \frac{(b_j - b_i)\,2^{11}}{[\Delta x/2]\,2^N}$$

N=# Left most zeros of Δ x up to 9
[Δ x /2] = upper 10 bits of Δ x

Figure 21 — Definition of a Segment

Edge-to-Segment Decoder

This unit is a pipeline device with eight levels, the first level receiving edges from the memory output data bus, and the seventh and eighth outputting one or two segments to the segment buffer for storage in a single word of memory (see Figure 22).

The inputs to the decoder are the data on the memory bus, < ΔY, M, X, S, B, G, C, F, V >, and Pipe Clock Enable (PCE) as well as a 134-nanosecond clock and buffer load Cycle Reset pulse (CRS) from the communications controller, where CRS marks the end of a buffer load cycle. The pipe is clocked at multiples of 134 nanoseconds, enabled by PCE from the memory controller.

Levels 1 through 6 convert edge data to segment data. Levels 7 and 8 buffer one or two segments for storage in one word of the segment buffer. Two consecutive segments must be stored together whenever the one on the left has a $\Delta x \leq 3$. A flag, e, is stored in each word of the segment buffer. If e is 0, it means that two segments are present, the

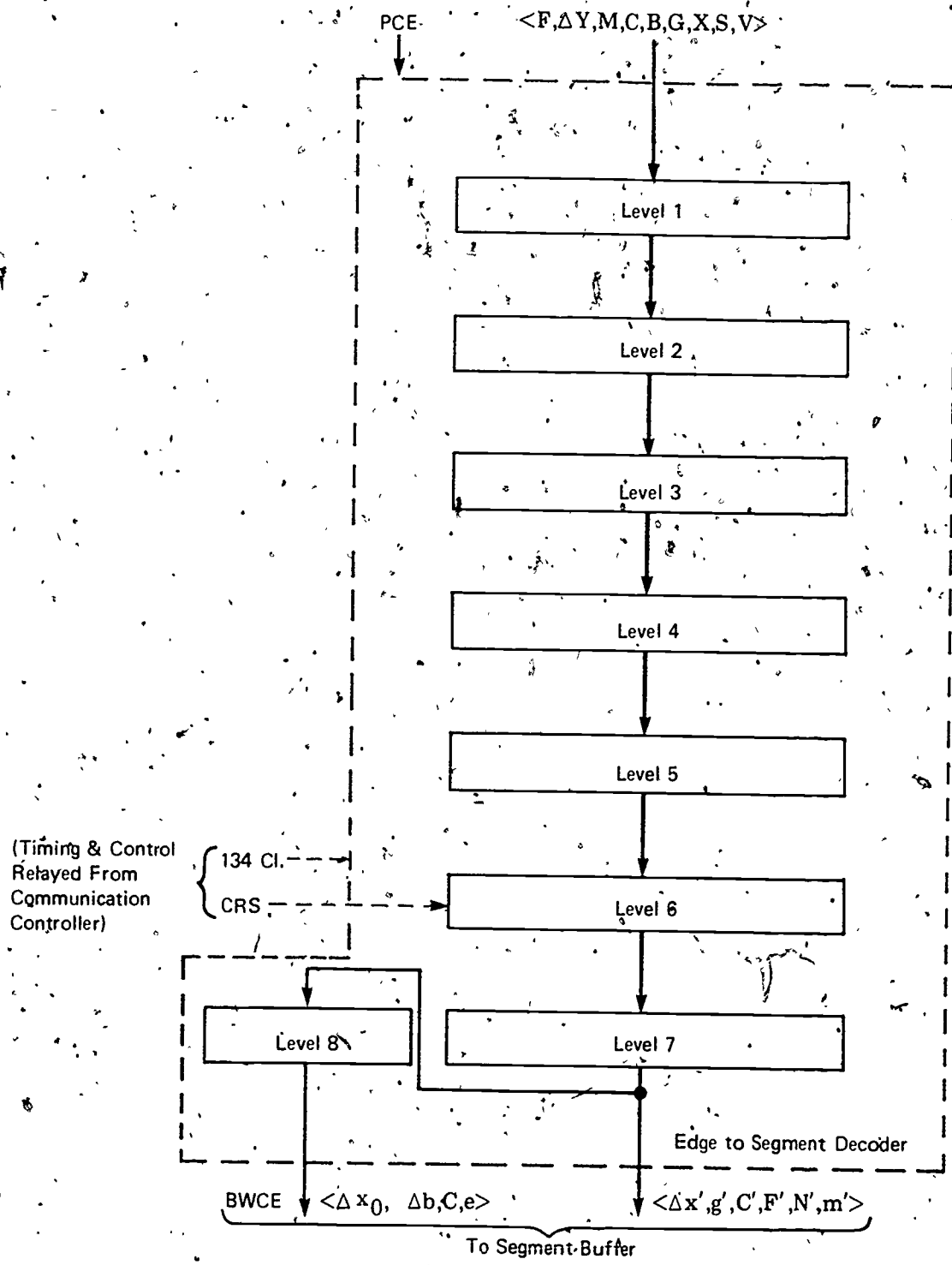


Figure 22 – Block Diagram of Edge to Segment Decoder

left one whose $\Delta x \leq 3$. Because this left segment's Δx is less than or equal to 3, only Δx_0 (its low order bit in Δx), an 8-bit Δb, and a 12-bit C are required to define that segment. The following right segment is defined by an 11-bit $\Delta x'$ a 3-bit N', a 10-bit g' a 1-bit F', a 12-bit C', and a 1-bit "first word" marker, m'. An m = 1 flags the word (when read by the video generator) as containing the first segment(s) of a scanline. A write enable, Buffer Write Clock Enable (BWCE), is generated by level 6 whenever data are present for storage.

Levels 1 and 2 (See Figure 23). Levels 1 and 2 generate the x and b intercepts for each edge passing through. Two levels are required because the multiplications involved cannot be done in one clock period of 134 nanoseconds. Level 1 receives as data input ΔY (= y - Y), M (exponent for S), S, X, B, G, C, F, and V (an edge validity bit). This level buffers these data where $\Delta Y \cdot 2^M$ is remembered rather than ΔY and M individually and produces as output intermediate products toward the solution for the intercepts.

Level 2 buffers the intermediate results and produces as output V (1 bit), C (12 bits), F (1 bit), x (11 bits), and b (7 bits) where:

$$x = \Delta Y \cdot 2^{M-11} \cdot S + X \text{ (rounded to an 11-bit integer)} \quad \text{(Equation 5)}$$

$$b = \Delta Y \cdot 2^{M-11} \cdot G + B \text{ (rounded to a 7-bit integer)} \quad \text{(Equation 6)}$$

The 134-nanosecond clock for these and following levels is enabled by PCE. The pipe's clock is inhibited, that is, PCE = 0, if the edge on the memory bus is invalid *and* is from the block of edges required for decoding the scanline being worked on. That is, an invalid edge will be placed upon the bus when all N edges entering the output register from a valid block of edges happen not to cross the decoded scanline. By inhibiting the clocking of the pipe decoder for invalid edges from a valid block, neighboring edges in the pipe will always be valid ones, will be in x order, and thus will be organized so that the lower levels of the pipe can form segment data out of neighboring edge data.

Until the first valid edge is placed upon the bus and after the last valid edge is placed upon the bus, the pipe is clocked at 134-nanosecond intervals independent of the validity of the data, guaranteeing that the block of valid edges that entered the pipe will pass through the pipe. The last valid edge that enters the pipe during a decode period, is followed by invalid data as it moves down the pipe. The segment to be generated to represent the scanline to the right of this last valid edge will be forced in level 6 to have a Δx (width) sufficient to guarantee that the scanline video to the right of this last valid edge's x intercept will extend all the way to the right on the screen. Of course, if this last edge is a smooth shaded edge (with F = 0), the video will be incorrect. In such a case

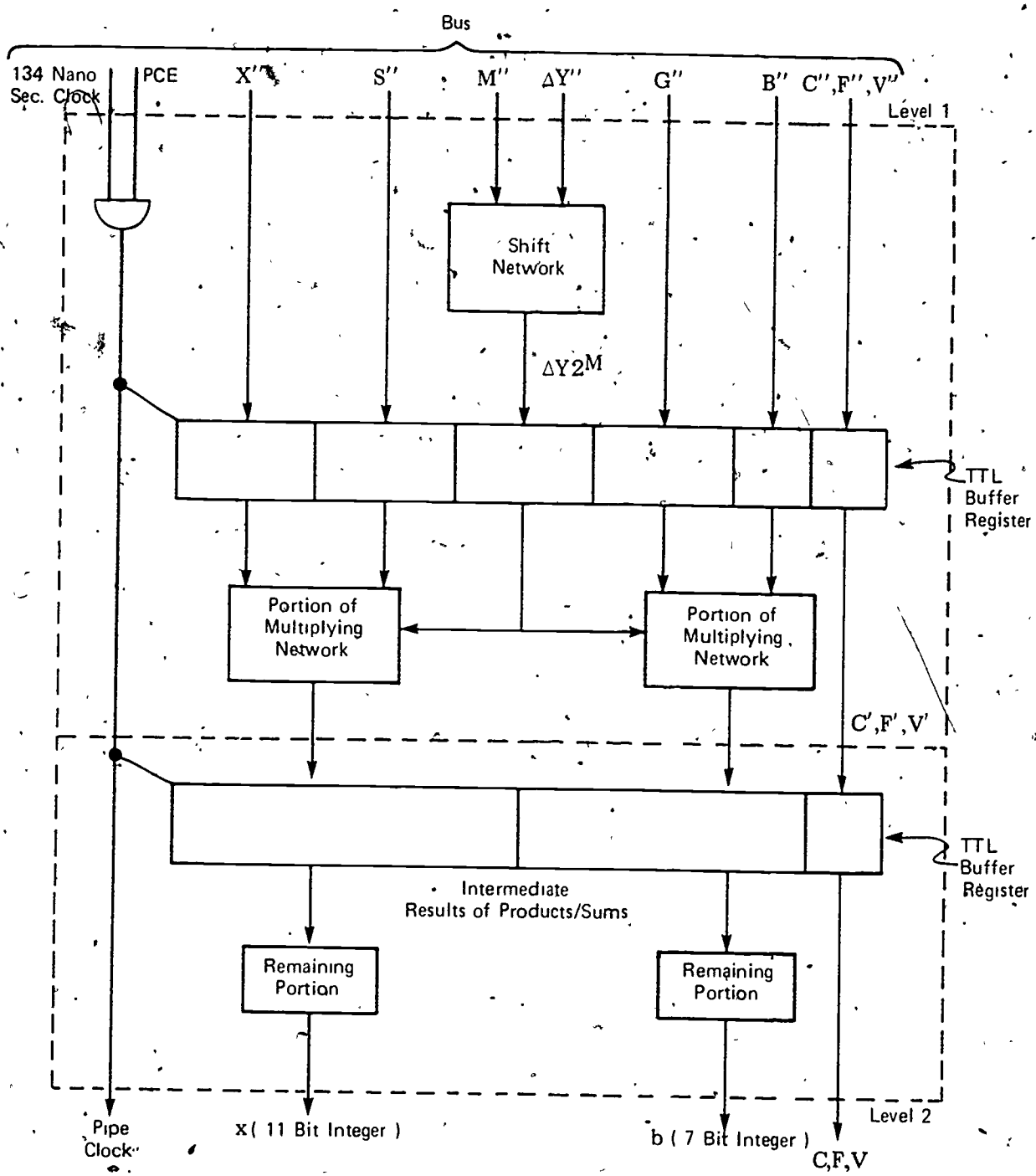


Figure 23 – Block Diagram for Levels 1 and 2

there should have been one more valid edge whose x intercept corresponds with the right edge of the picture (at x = 1600).

For notation convenience, primed variables shall refer to those of the next edge. Thus, for a scanline, if B refers to the brightness of an edge i, then B′, B″, and so forth, refer to the brightness of the next and following edges just to the right of edge i.

Levels 3 and 4 (See Figure 24). Level 3 buffers the data from level 2. Level 4 buffers the data from level 3 and produces as output V, C, F, Δx (11 bits), and Δb (8 bits, 7-bit magnitude plus a sign bit) where:

$\Delta x = x' - x$ (x′ is the x intercept of the following edge) (Equation 7)

$\Delta b = b' - b$ (b′ is the b intercept of the next edge) (Equation 8)

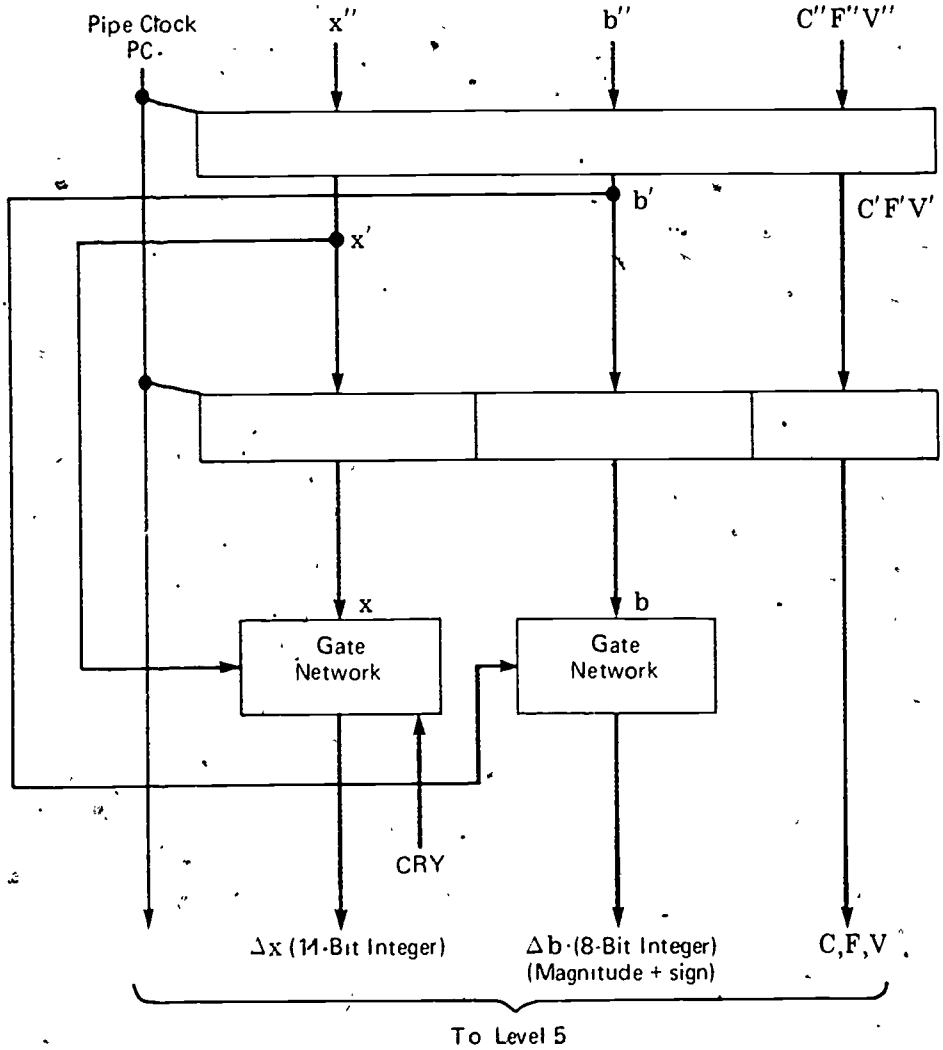


Figure 24 – Block Diagram of Levels 3 and 4

A carry bit, CRY, which reduces Δx by one in this level 4, is true if the previous Δx, now in level 5, is less than 2 (Δx in level 5 are forced up to a value of 2 if they are 1 or 0 in level 4).

Level 5 (See Figure 25). Level 5 buffers its input from level 4 and produces as output Δx, Δb, r (10 bits), F, C, V, Less Than Four (LTF), and Valid Not Last (VNL). This level relays on Δb, F, C, and V. LTF is true if Δx is less than four. If Δx is less than 2, Δx is set equal to 2 and CRY is generated, forcing the $\Delta x'$ of the next edge to be reduced by one unit.

Thus, Δx, if not greater than 1, is forced equal to 2. The Δx for the following edge will then be reduced by 1 and any remainder remembered in the "correction register." If this next edge's Δx is then also too small, it too is set to 2, forcing still the reduction of the next edge's Δx until some Δx can absorb the correction. Because edges will be generated so as to be two units apart after x is truncated to 11 bits but before S is truncated to 13 bits, the corrections to Δx in the pipe will never be required to move an x intercept more than one unit in x and only when two neighboring edge x intercepts happen to round off to a one-unit separation. Δxs are forced to a minimum of two units in order that the segment-to-video decoder unit need not do register-to-register transfers at clock intervals less than 67 nanoseconds (equivalent to two units in x along a scanline) and in order that the segment buffer does not clock at intervals less than 134 nanoseconds.

Δb is in a magnitude-plus-sign form because it shall be used by a multiplier in a following layer and because the segment to video decoder requires that form.

The 10-bit positive integer, r (no sign bit), is defined by the equation for F = 0 segments

$$r = \frac{2^{18}}{[\Delta x/2]\,2^N} \text{ (rounded off to a 10-bit positive integer which is } \leq 512) \qquad \text{(Equation 9)}$$

where N is the number of consecutive leftmost zeros in Δx up to nine and where $[\Delta x/2]$ is an integer formed by the upper 10 bits of the 11-bit integer Δx (i.e., truncate to an integer, [ ], the number $\Delta x/2$).

For F = 1, r is set equal to 2^9, the value in the above equation where $[\Delta x/2] = 1$. Since r will be multiplied times Δb in the next levels (6 and 7) to form a brightness gradient g, an r of 2^9 will guarantee Δb can be recovered from g later in the segment-to-video decoder via a shift network rather than via an expensive divider network (actually $g = 2^2 \Delta b$ when F = 1). Thus, for F = 1 cases, where g must not be used to alter

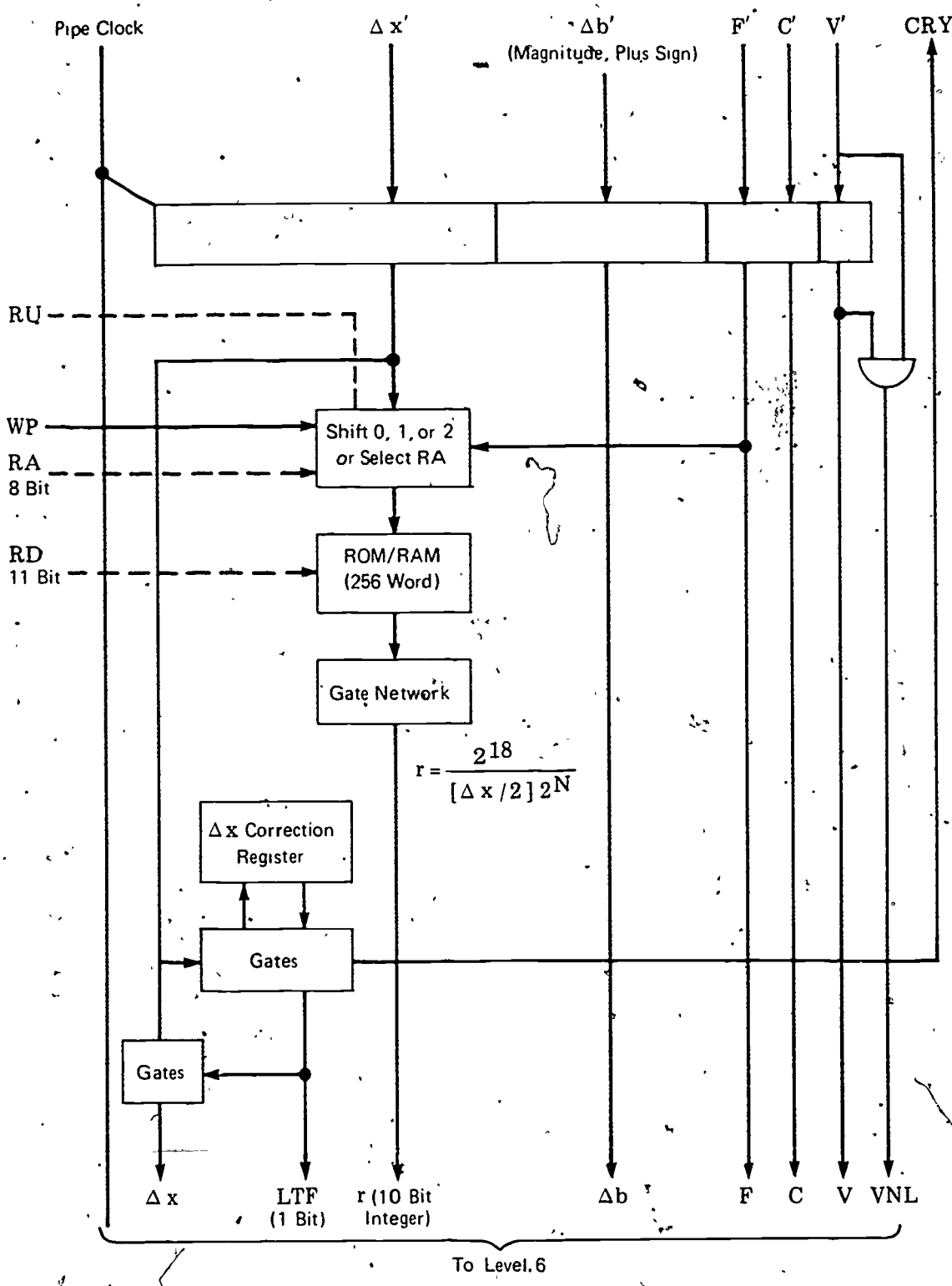


Figure 25 – **Block Diagram of Level 5**

the brightness (b) along a scanline over the segment's range in x until the end of the segment, the brightness must jump by Δb. This will be discussed further when the segment-to-video decoder is described.

The variable r is obtained by a Read Only Memory (ROM) (or Random Access Memory, RAM) table look-up. The function $[\Delta x/2]$ behaves in such a way for each increment in Δx, that a 256 11-bit word table can be employed, consisting of an 8-bit "1/x" mantissa and 3 correction bits.

To use the table, first μ (up to 2 0s) are shifted out of $[\Delta x/2]$ if possible. Out of the resultant number $A(a_9 \ldots a_0)$ the upper eight bits $(a_9 \ldots a_2)$ are used to look up the value of

$$T = \frac{2^{18}}{2^{N-\mu}\ [a_9\ \ldots\ a_2\ 1\ 1]} - 1 \qquad \text{(Equation 10)}$$

This denominator differs from $\frac{1}{2^N[\Delta x/2]}$ only in that two 1s reside in the lower 2 bits. If $a_1 = a_0 = 1$, the correction bits are ignored and a 1 is added (cancels the -1 in Equation 10). If $a_1 = 1$ and $a_0 = 0$, the first correction bit plus 1 are added to T. If $a_1 = 0$ and $a_0 = 1$, the first and second correction bits plus 1 are added to T. If $a_1 = a_2 = 0$, all the correction bits plus 1 are added to T. Equation 10 employs a "-1" in order to make the 9th bit of T a 1 and the 10th bit 0 for all cases. Thus, the 9th and 10th bits of r need not be stored in the table. After corrections, a 10-bit integer results whose maximum value is 512.

When a RAM is employed, RAM Data (RD) can be written at address RA (RAM Address) upon enabling (selecting) RA with RU and strobing in the data with a Write Pulse (WP). RA, RU, RD, and WP are generated by the terminal control unit and communication controller. RA and RD are placed upon the memory input data bus on these occasions.

Levels 6, 7, and 8 (See Figure 26). These last three levels obtain the brightness gradient for each segment, buffering up to two segments for storage in a single word of the segment buffer.

Level 6 receives as input V, Δx, r, Δb, C, F, VNL, LTF, and CRS, and buffers all but VNL and CRS in a TTL register. If VNL is true, the segment is valid and not last, in which case Δx is not replaced by Δx_{max} when buffered in the TTL register. (See the select network in Figure 26.)

A multiplier network distributed in level 6 and level 7 forms the product where:

$$g = \Delta b \cdot r \cdot 2^{-7}\ (\pm 1/2) \qquad \text{(Equation 11)}$$

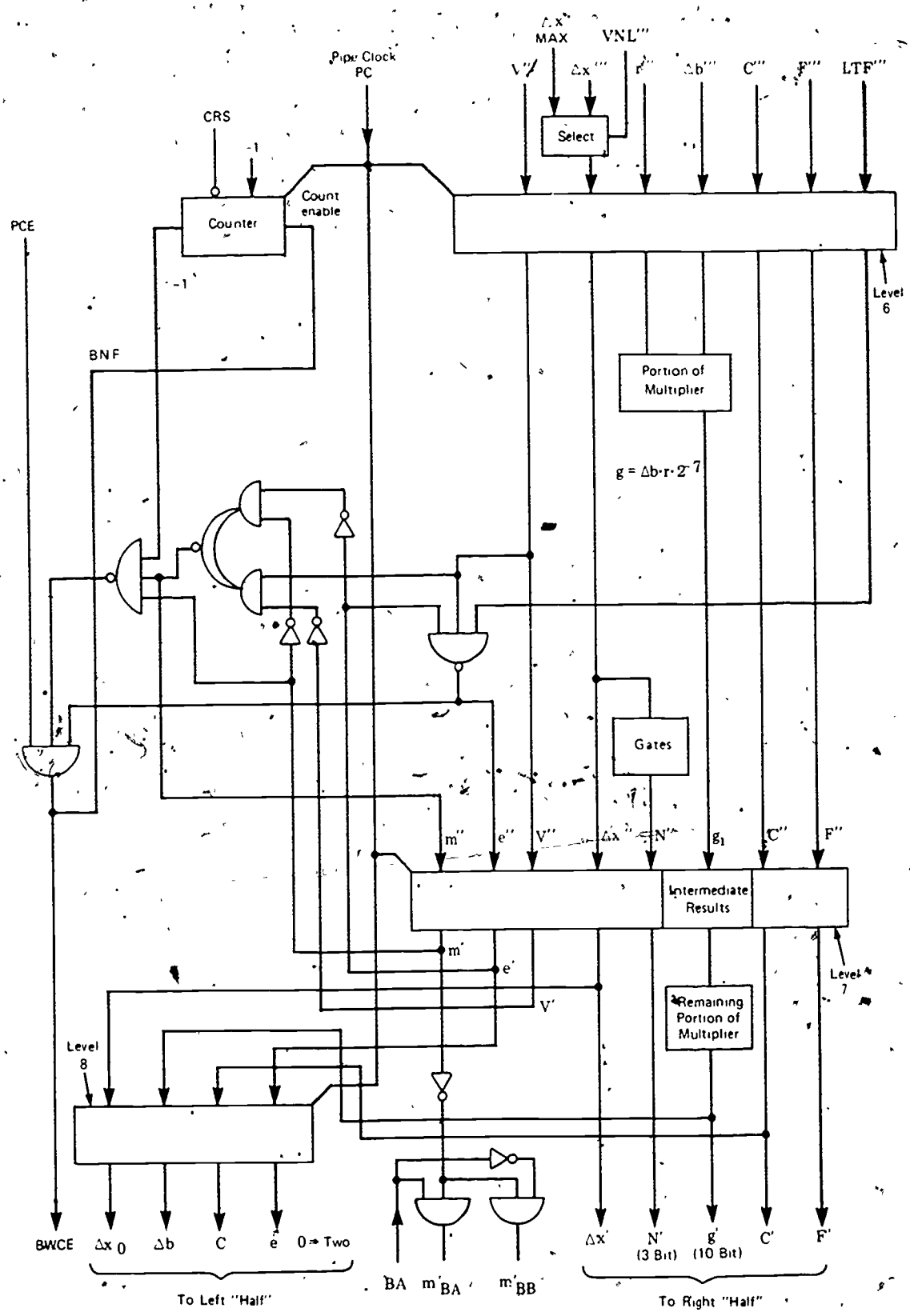


Figure 26 – **Block Diagram of Level 6, 7, 8**

a 10-bit signed (magnitude plus sign) integer that is rounded off to the nearest integer. The number N of consecutive leftmost 0s up to 9 in Δx is formed by a gate network. Both g_i and N, along with V, Δx, C, and F are buffered in level 7 where a ripple carry can be completed in the formation of g, and where a segment can reside during a write cycle of the segment buffer.

Normally, segments with $\Delta x \geq 4$ are stored in the segment buffer while in level 7. If a segment's $\Delta x \leq 3$ and that segment follows a segment stored while in level 7, the storage clock enabling signal e is held false so that when that segment enters level 7's TTL register, it will not be stored until it reaches level 8.

Level 8, which buffers Δx_0, Δb (equal to $g \cdot 2^{-2}$), C, and e for a segment. The variables e and Δx_0 are sufficient to define Δx and to flag the segment as valid. That is, if e = 1, in level 8, it means it had been stored in level 7; therefore, it is not valid in 8. If the following segment in level 7 is being stored, the data in level 8 thus will be flagged as invalid (i.e., already stored). If e = 0 in level 8, it means Δx was less than 4, that is, $\Delta x = 2$ or 3, in which case Δx_0 (the low-order bit of Δx) is sufficient to define Δx. Because a segment with a $\Delta x \leq 3$ has $[\Delta x/2] = 1$, no smooth shading through intermediate brightnesses is necessary. Thus F is not needed.

The Buffer Write Enable e is "anded" with PCE and Buffer Not Full (BNF) to form BWCE which causes the segment buffer to store the word. PCE is required so that each segment is stored only once. BNF is required to prevent overflow in the segment buffer in cases where more than 256 edges are decoded (error in image definition). Each time a word is stored in the segment buffer, a counter is incremented (from -1, then to -256, -255 . . . to -1). Initially this counter is at -1 (forced to -1 by CRS at the beginning of the buffer load cycle) or having been left at -1 during the previous load cycle during which 256 "writes" were made. Often, there is not enough time and/or edge crossings during a load cycle to write 256 words into the buffer (thus leaving the counter below -1). BNF goes false if the counter = -1 and the segment is not the first valid one encountered during a load cycle. The first valid segment where m = 1 from level 6 forces BNF true in spite of the -1 value, resetting the counter to [redacted] where it will be incremented by following valid data. If the first valid segment has a $\Delta x \leq 3$, even though BNF goes true, e does not, preventing the storage of this segment while in level 7. In such cases, m is set to 1 for the next segment also. Thus, when the pair of segments is stored, m will be 1 in the first word to enter the segment buffer (an e of 0 will guarantee that e is 1 during the next clock period).

A segment is detected as "first" if it is valid (V = 1) and the previous segment was invalid (V = 0) ($V' \cdot \overline{V}''$ in the diagram). The memory controller is designed to set V equal to zero for all but valid edges except if none are encountered. That is, if it happens that no valid edges (including the marker edge) are detected during a scanline decode period, the last invalid segment is flagged as valid, in order that the segment buffer will store a word with m = 1. This is necessary in order that the segment to video decoder unit can find the first segment in the buffer when less than 256 words are loaded into the buffer.

When V = 0 and PCE is 1 following the block of valid segments that have passed down the decoder through level 7, BWCE is held high until (a) the buffer fills with "nulls" or (b) the load cycle times out. A "null" is simply that which follows segments that define the visible scanline. In cases where two valid segments enter each word of the segment buffer, only 128 words will have been written in, yet another 128 shifts are required to move the segment data to the output of the shift register buffer. Thus, it is necessary to use up any remaining cycles of the load cycle (40 of them) to move these data forward as far as possible, minimizing the number of shifts that must be done by the buffer to video decoder during the horizontal retrace period before an m = 1 is detected and the visible trace begins.

When one-half of the segment buffer (which is actually a pair of ping-pong buffers) is switched over to the video decoder at the end of a load cycle, m is forced low to that half of the ping-pong buffer so that when an m = 1 is loaded into the other half of the buffer, it does not enter the first half also. This "extra" m = 1, if not shifted out during a load cycle which "times out" because several segments are paired into cycle words, would lock the video decoder onto the wrong "first word."

It should be remembered that for segments entering level 8 as valid, $2^{-2} \cdot g = \Delta b$ (since $[\Delta x/2] = 1$), and that segments where F = 1 have $g = \Delta b.2^2$ also (that is, Δb is upper 8 bits of 10-bit gs).

Segment Buffer

The segment buffer consists of two ping-pong shift register buffers feeding one output register. The 60-bit data word from levels 7 and 8 is fed to both buffers. During a scanline decode period, one buffer is loaded while the other is unloaded through the output register, the roles of each buffer alternating every scan period. The buffer being loaded is clocked (shifted) at the end of the 134-nanosecond period when BWCE = 1. The buffer being unloaded is under the control of the segment-to-video decoder yet to be described. However, m to a buffer being unloaded is held to 0 to guarantee no false "first" segments enter.

At the beginning of a load cycle which begins six clock periods (6.134 nanoseconds) after a scanline decode period because of the delay for a new set of segments to pass down the pipe, the counter in level 6 already has been reset (by CRS) and a buffer load/unload select flipflop (BA) has just been switched to the opposite state. Each word written into the segment buffer causes the counter in level 6 to be incremented (enabled by BWCE). After the last valid segment is dropped in levels 7 or 8 (i.e., following Vs = 0), BWCE remains high (so that the buffer will be clocked steadily at peak rate 134-nanoseconds interval) until either it fills (counter has cycled) or it is time for a new load cycle. At the end of a load-unload cycle, the roles of the two buffers are interchanged.

The segment-to-video decoder via Buffer Read Clock Enable (BRCE) clocks the buffer (to be unloaded) once and if m = 0 in the buffer output register continues to clock the buffer until the marker bit, m, is detected as equal to 1 indicating that the data in the buffer have been shifted forward and now reside in the output register of the buffer. During this "search" clocking, m entering the buffer is forced to 0 (see Figure 27). If, as a worst case, two valid segments, for instance, were loaded into each word for a total of 128 words of the buffer, and if there was not sufficient time for clocking the buffer another 128 times before a new load cycle, the segment data in the buffer to be unloaded will not have been shifted forward to the output of the shift register buffer. Thus, this partially loaded buffer must be clocked by the video decoder unit until the first valid segment has been loaded into the output register. The video decoder takes over the control of a buffer at the beginning of a horizontal retrace period and thus has time to shift forward the data.

The main memory requires 450 clock cycles (for N = 4 at intervals of 134 nanoseconds), so that the segment buffer will require 450 clock cycles to elapse during a load cycle before it can begin to shift forward its contents. An additional 40 clock cycles for a scanline decode period will guarantee that at least 128 + 40 (or 168) words have entered the buffer by the time the unload cycle ends. Thus, during retrace time, the buffer will have to be shifted by the video decoder at the most, only 256—168 or 88 times +2 more in order to get data into buffer output registers. Thus, at 134-nanosecond clock rate, 90 x 134 = 12.1 microseconds are required. Because the visible trace time is 400 clock periods and the scan period has to be equal to a multiple of 5 (for a 5:1 interlace), the trace period of 90 is used to bring the scan period to 490 (i.e., 98 x 5). Thus, the retrace time is 12.1 microseconds (90 x 134 nanoseconds). The visible scan period is that required to output 1600 points or 400 times 134 nanoseconds, which

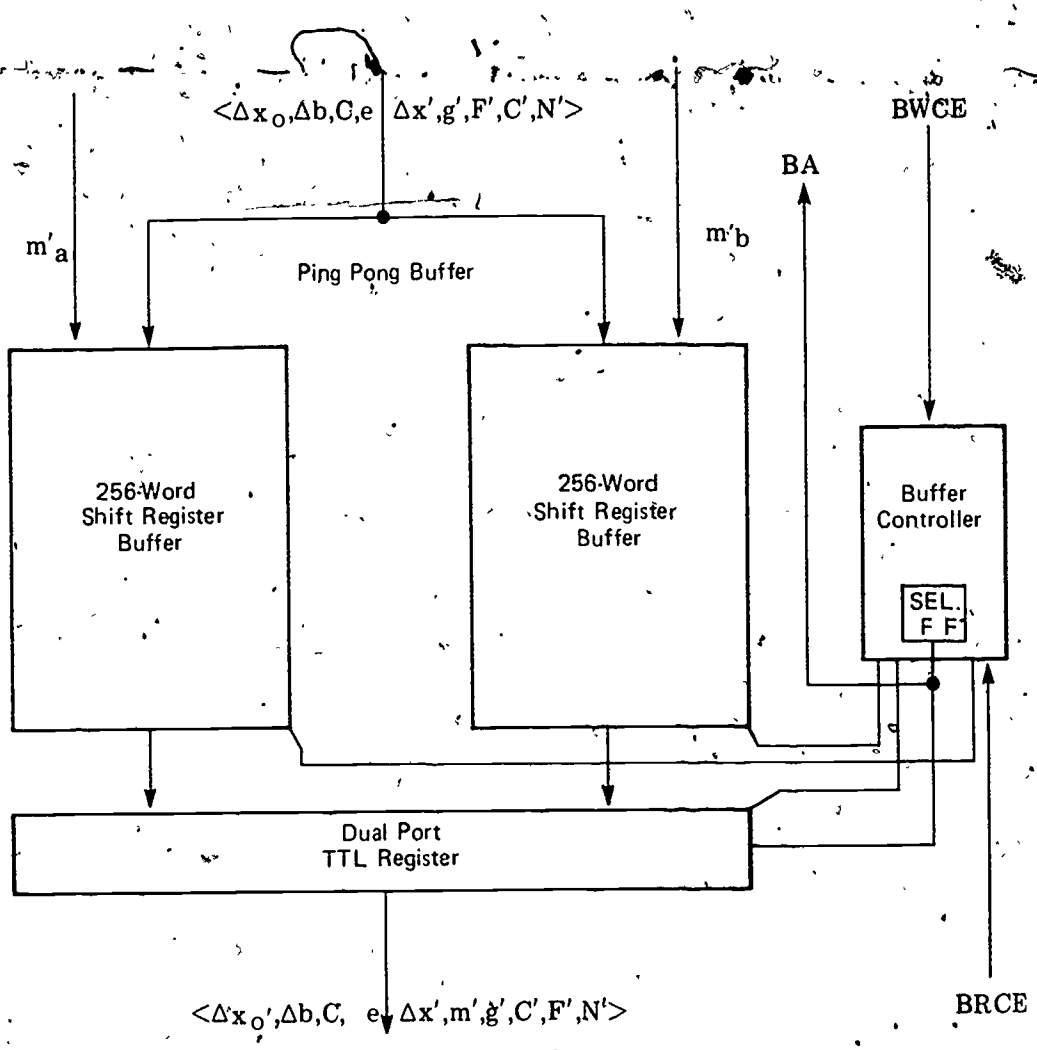


Figure 27 — **Block Diagram of Segment Buffer**

comes to 53.6 microseconds for a total scanline period of 53.6 + 12.1 or 65.7, that is, 400 + 90 clock cycles of which 490 clock cycles are required to decode the data.

A marker segment (generated out of marker edge) should be the first edge to enter the segment buffer. Its Δb assumes B = 0 on its left. Thus, its Δb will be equal to the B of the next edge down the pipe.

A scanline period of 65.7 microseconds is, for all practical purposes, the same as that for commercial TV (which is around 64 microseconds). Thus, the display monitor will remain compatible with commercial TV input (see Figure 28.)

Segment-to-Video Decoder

There are four major registers in the video decoder. They are the Color Output Register (CR), and Brightness Register (BR), the Brightness Increment Register (BIR),

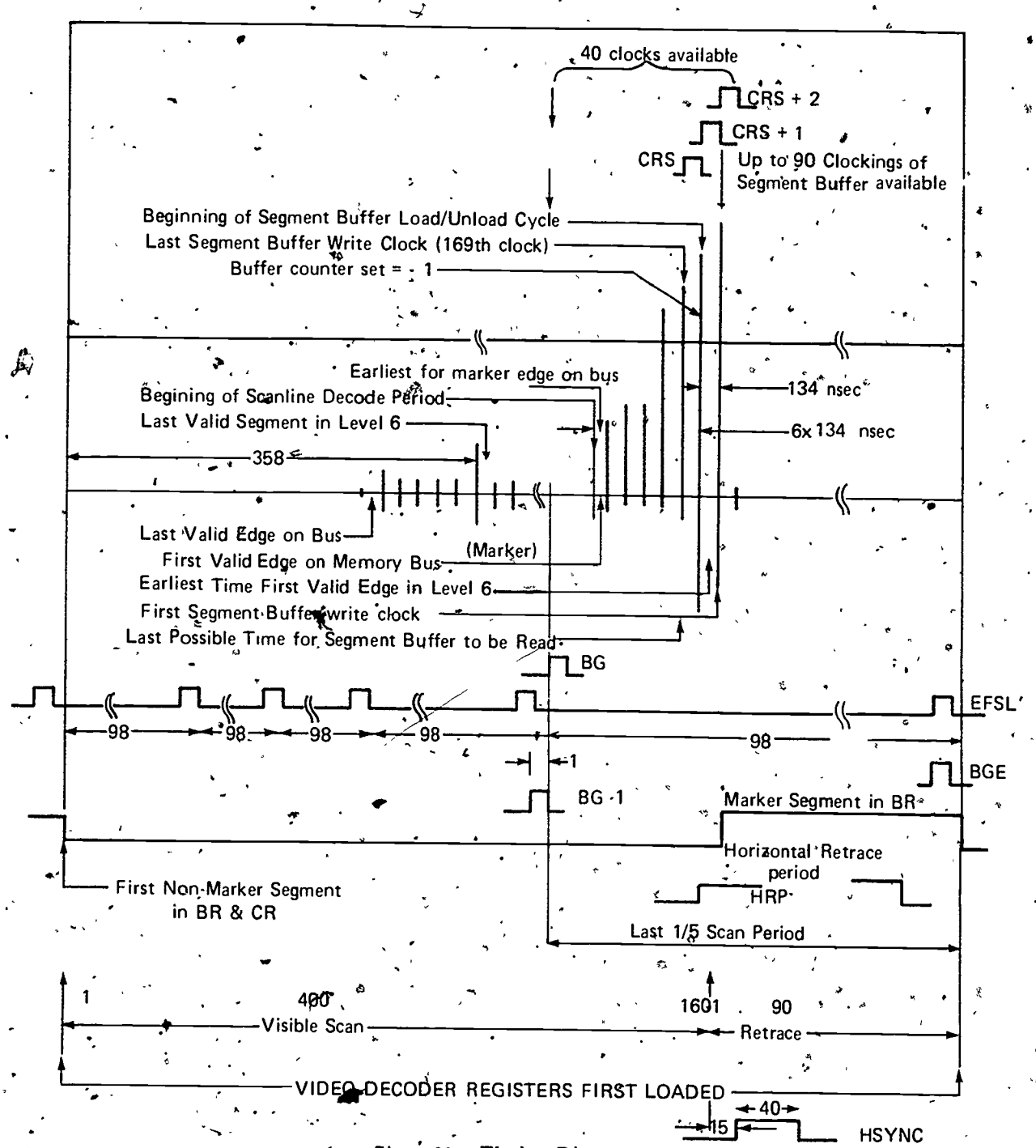


Figure 28 – **Timing Diagram**

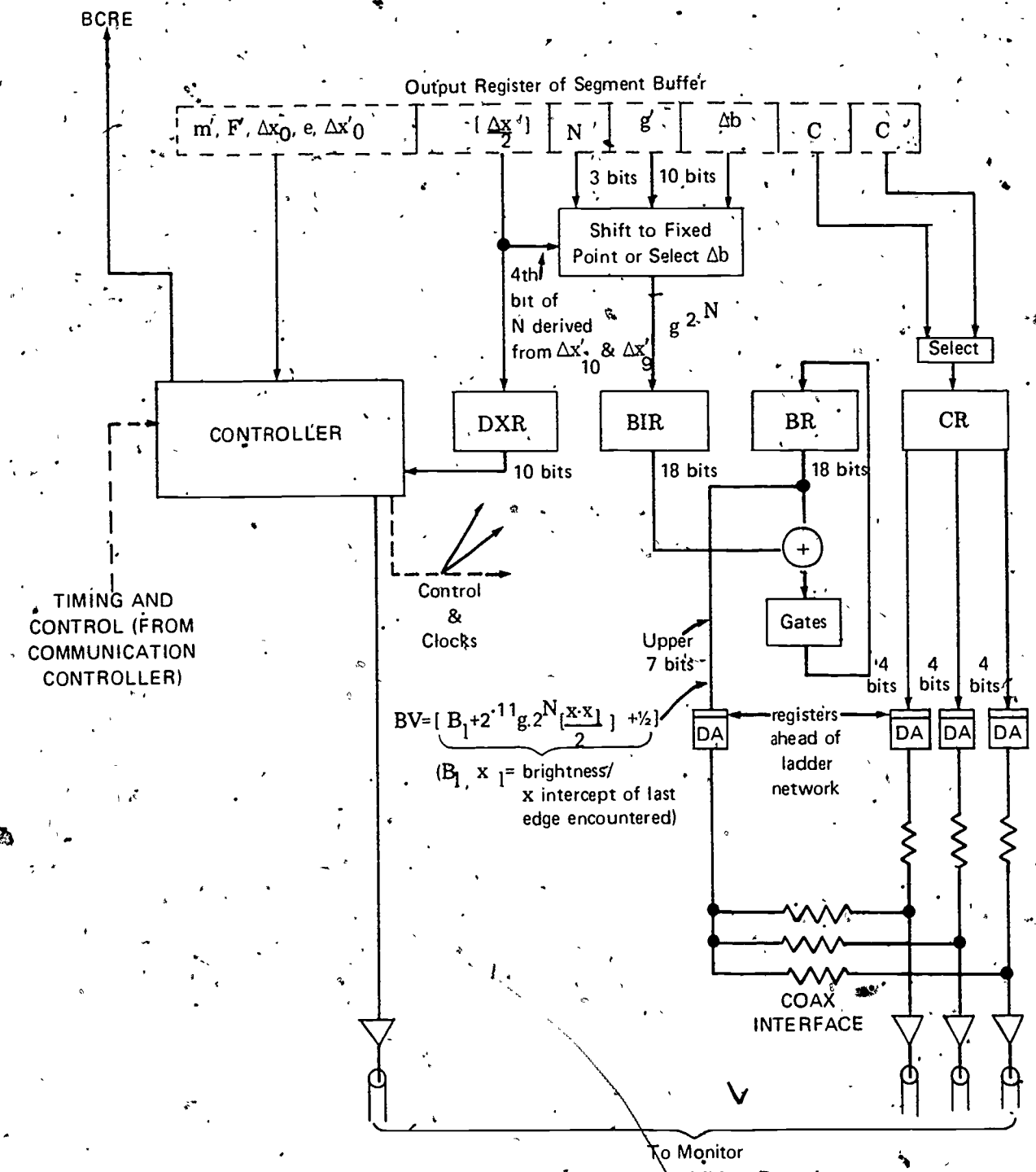


Figure 29 – **Block Diagram of Segment to Video Decoder**

and a Δx Counting Register, DXR (see Figure 29). In addition to these registers, there is a set of registers in each Digital to Analogue (DA) converter network. However, these need not be considered here.

Most of the data for a segment resides in the BIR, CR, and DXR registers for exactly $[\Delta x/2]$ clock periods, all of which are 67 nanoseconds long except the last period, which must be 100.5 nanoseconds long if Δx is odd (see Figure 30).[1] m', F', e, Δx_0 and $\Delta x'_0$ are left in the Segment Buffer Output Register (SBOR). The BIR is loaded either (a) with g' approximately shifted by N' and high order bits of $\Delta x'$ (N' is held to 3 bits with high bits of $\Delta x'$ used to derive the 4th bit later), or (b) with $\Delta b \cdot 2^2$ of a left (narrow) segment. The contents of the BIR are used to increment the brightness register whose upper 7 bits define the brightness of the video to be sent to the monitor. Both the brightness and brightness increment registers are 18 bits wide.

For right segments, if F is 0, the contents of the BIR are added to that of the B register, the updated B being clocked back into the B register at the end of each of the $[\Delta x'/2]$ clock periods (see Figure 30).

F is 1 if for the right segment the contents of the BIR (where $g' = 2^2 \Delta b'$) are added to the contents of the B register, the updated B being clocked back into the B register *only* at the end of the *last* clock period of a segment's decoding (see, in Figure 30, the interval between $X_3 + X_4$).

For left segments the contents of the BIR (containing Δb) are added to that of the B register, the updated B being clocked back into the B register at the end of the one and only one clock period over which that segment is valid (remember only segments with $\Delta x = 2$ or 3 will be left segments where $[\Delta x/2] = 1$ (see, in Figure 30, the interval between $X_2 + X_3$). When new segment data are loaded into the BIR, DXR, and CR registers, the BR register is loaded with its input truncated to an 8-bit number, the lower of the 8 bits being set to 1 and the remaining 10 bits being set to 0. This prevents the accumulation of round-off errors stemming from the generation and truncation of g to a 10-bit number, and effectively rounds off the output 7 bits of B to the nearest integer.

The DXR register is loaded with $[\Delta x'/2]$ data only from the right segment, the only segment that can have a $\Delta x > 3$. After having been loaded from the right segment, this counter counts down and, as it approaches 0, enables various clocks. For example, the segment buffer memory is read (clocked) always at the beginning of the last 67- or 100.5-nanosecond period over which the right segment is valid.

[1]The extended clock period occurs in the last period, during which a segment's data are in the registers of the DA units.

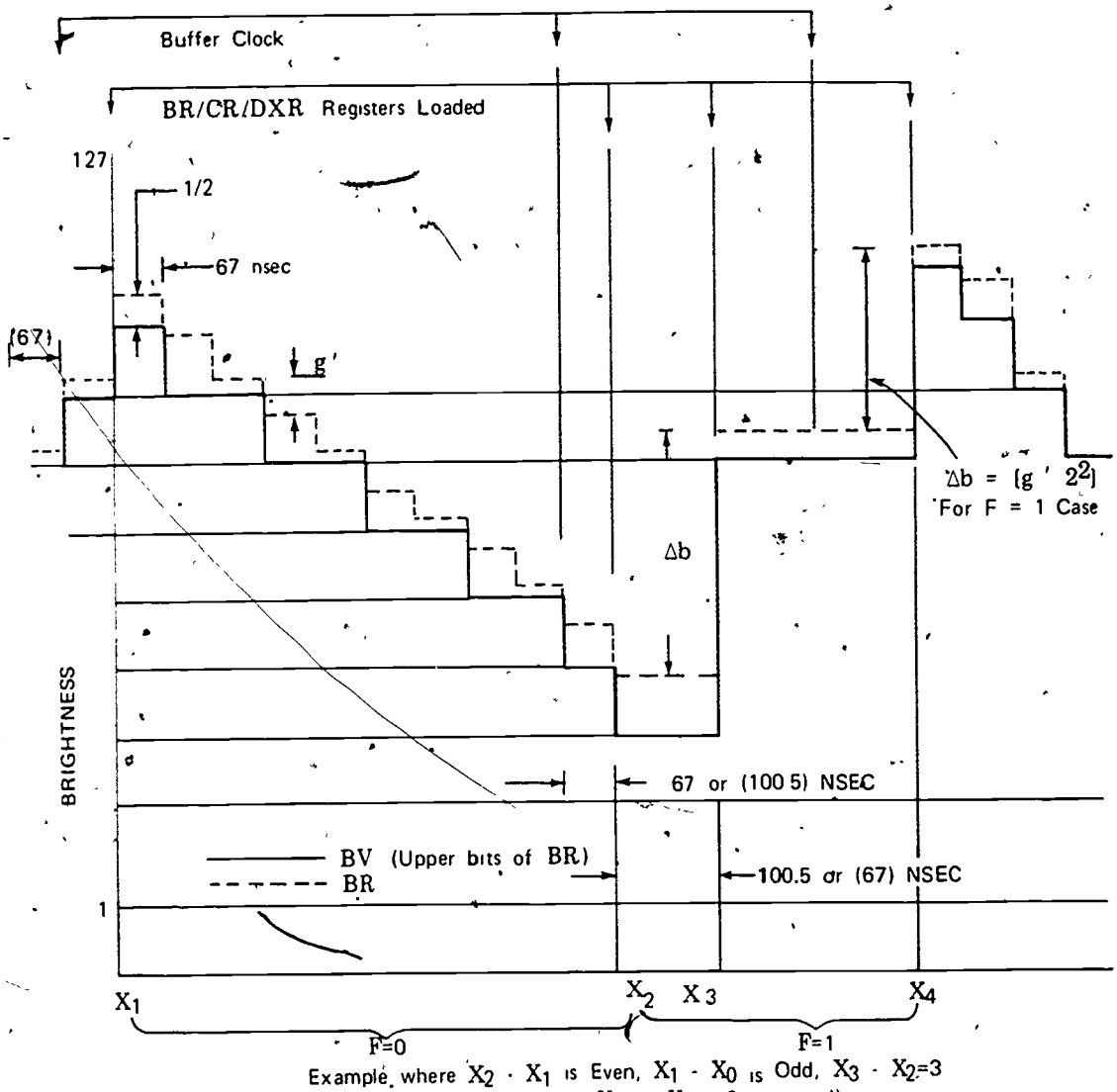


Figure 30 – **Brightness Decoding**

For an odd Δx, the clock intervals of 67 nanoseconds will be extended to 100.5 nanoseconds by the video decoder controller during the last clock period where segments' data resides in the DA converter registers.

A controller section of the video decoder contains a register used for load selecting and clock enabling of the other registers.

The outputs of the BR and CR registers pass through four D-A converters (with hold registers) where the analog outputs are added together by simple resistor networks, as shown in Figure 29. The three color video signals (red, blue, and green) are fed onto

the color monitor through coax drivers. Sync is obtained directly from the terminals communication controller unit, which supplies the basic timing and control signals for run and load modes.

BRIGHTNESS DECODE ERROR ANALYSIS

Because segment data use a gradient approach that could accumulate truncation errors if insufficient bits are employed, an error analysis is carried out next.

The formulation presented in describing the terminal always treated data as integers. Thus, various powers of 2 were injected into the various equations. This was done to simplify this error analysis.

Substituting Equation 9 in Equation 11, it follows that:

$$g = \Delta b\, r \,.\, 2^{-7} \pm 1/2 = \frac{\Delta b 2^{+11}}{[\Delta x/2]2^N} \pm \Delta b 2^{-8} \pm 1/2 \qquad \text{(Equation 12)}$$

But BIR is $g \,.\, 2^N$.

Thus, when BR is truncated to $B_\ell + 1/2$ (assuming radix point to the right of the 7 most significant bits in BR), the following change in brightness, db, after $[\Delta x/2]$ brightness update results:

$$db = \left[\frac{\Delta x}{2}\right] g\, 2^N 2^{-11}$$

$$= [\Delta x/2]\ 2^{N-11} \left\{ \frac{\Delta b 2^{11}}{[\Delta x/2]2^N} \pm \Delta b 2^{-8} \pm 1/2 \right\}$$

$$= \Delta b \pm \left[\frac{\Delta x}{2}\right]\ 2^{N-19} \Delta b \pm \left[\frac{\Delta x}{2}\right] 2^{N-12}$$

$$= \Delta b \pm e$$

where:

$$e = \left\lceil \frac{\Delta x}{2} \right\rceil\ 2^{N-19} (\Delta b + 2^7)$$

Since $\left[\frac{\Delta x}{2}\right] \,.\, 2^N < 2^{10} - 1$ (a 10-bit integer) and $\Delta b < 2^7 - 1$ (a 7-bit integer), it follows that:

$$e < (2^{10} - 1)\, 2^{-19} \,.\, [(2^7 - 1) + 2^7]$$
$$< (2^{18} + 1 - 2^8 - 2^{10}).\ 2^{-19}$$
$$< 2^{-1} - 2^{-9}$$

The 18-bit BR can resolve to 2^{-11}. Thus, BR just before truncation will contain

$$B_\ell + 1/2 + \Delta b \pm e \text{ where } e < 1/2 - 1/2^9 \qquad \text{(Equation 14)}$$

It follows that the maximum possible contents at truncation will be $B_\ell + \Delta b + (1 - 1/2^9)$, which truncates $B_\ell + \Delta b$, and the minimum possible contents will be $B_\ell + \Delta b + 1/2^9$, which also truncates to $B_\ell + \Delta b$. Thus, via the truncation technique employed, each edge's brightness intercept is restored at the beginning of the associated segment's arrival in BR.

COLOR MONITORS

A video amplifier rise time of about 70 nanoseconds is required to follow the output of the D/A networks of the terminal decoder. This can be accomplished within a Sony commercial color TV set by (a) reducing the value of the video output resistors to 2K ohms, and (b) either lowering the supply voltage to 75 volts or using higher power transistors.[1]

Since D/A outputs are the log functions of the required red, blue, and green video signals that must be fed to the color picture tube, each of the three video amplifiers must be preceded by an antilog amplifier.

An antilog circuit with output response time of 60 nanoseconds can be realized by using the log characteristics of a transistor. However, a slow responding biasing feedback network is necessary in order to stabilize the amplifiers over the varying temperatures produced by the varying video signal levels. This feedback (via biasing the antilog transistor) clamps the color signal output on an antilog amplifier during horizontal retrace time to a fixed level. This feedback bias is additive with respect to the log color input so that the effect of the feedback is to set and to stabilize the contrast of the image on the face of the monitor's CRT. The brightness control serves to add a constant to the final color video signals. A contrast control is obtained by letting it control the clamping level during retrace time.

Both horizontal sync and vertical sync are fed directly into the appropriate places disabling the internal sources of sync within the TV monitor.

No attempt has been made to code the sync and color onto a single wire. Many techniques are available and can be employed when and if necessary.

[1] Identification of products in this report is for research documentation purposes only, this listing does not constitute an official endorsement by either HumRRO or the Department of the Army.

COMMUNICATION CONTROLLER

The communication controller consists of three major units: a coax receiver unit, a timing generator unit, and a terminal load control unit (see Figure 31). The coax receiver (a) derives both clock and data sync out of the synchronous data bit train which is sent over coax from the central system, and (b) produces as output an 80-bit data word to the terminal load control unit.

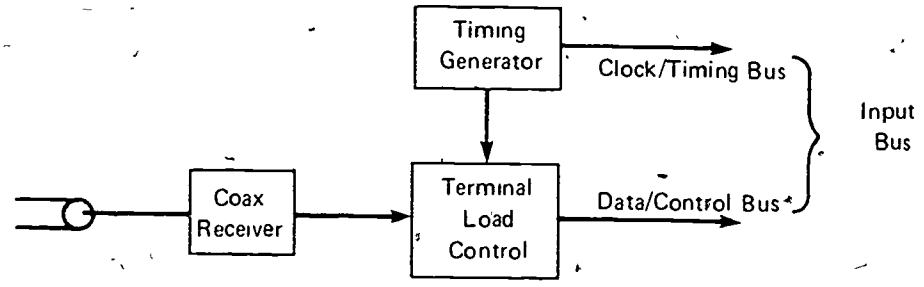


Figure 31 – The Communication Controller

The terminal load control unit decodes the 80-bit words from the coax receiver as commands or edge data (or as RAM data) and produces as output to the terminals' input bus (data/timing/control bus) the appropriate signals required to load or modify the terminals' memories. A normal load sequence consists of an attention command, followed by address and data commands, then an end attention command.

The timing generator unit generates a 34-nanosecond clock from which it derives all terminal clocks and timing signals including horizontal sync and vertical sync for a 5-to-1 interlaced raster scan, with a frame rate of 12/sec and a field rate of 60/sec (a frame is made up of 5 fields).

Coax Receiver

Synchronous transmission is employed, using pulse width modulation. An 84-bit word is required—1 bit for sync, 2 bits for communication control, 1 bit for parity, and 80 bits for commands or edge data. This is illustrated in Figure 32.

When no valid data are being sent down the coax, a null word is sent. It consists of a sync bit followed by at least two 0s (command bits to the communication controller, not to the terminal). If the parity bit and the following 80 data bits in a null word are also set to zero, the coax receiver derives sync information from the pulse train by locking onto the sync pulse following 83 consecutive 0s. The coax receiver is shown in Figure 33.

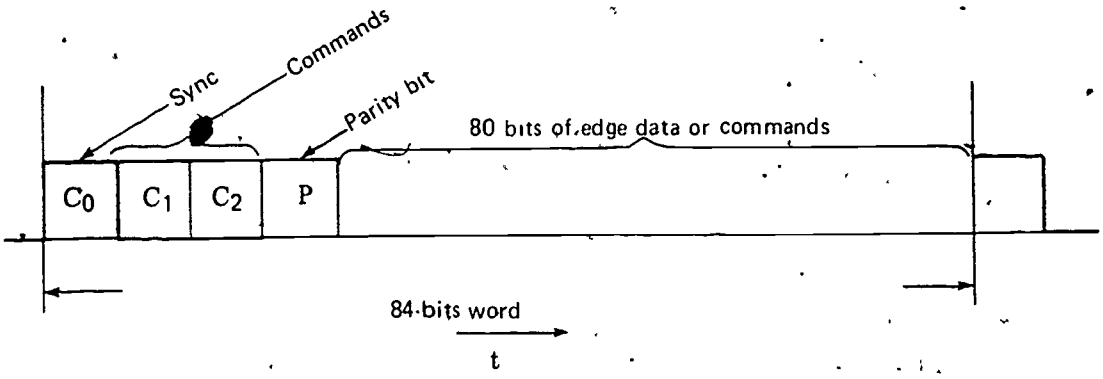


Figure 32 – Coax Data Format

Each bit transmitted consists of a high of τ units followed by a low of τ units, followed by the data bit of 2τ units. The negative transitions in the pulse train delayed by 2τ units serves as the clock for circuits which (a) sample the data train (see Figure 33), (b) derive sync information, and (c) convert serial data to parallel data.

The clock derived from the data train increments both a sync counter and a data bit counter, both cycling through 84 states.

The sync counter is always reset to -84 if a nonzero bit is detected. This counter will reach -1 only if 83 consecutive zeros have been detected. A "1" data bit "anded" with the -1 from the sync counter serves to synchronize the data bit counter to the sync pulse that follows the 83 zeros.

The data bit counter generates timing information flagging the end of every 4th bit in the 84-bit word and the end of every 16th bit in the latter 80-bit portion of the 84-bit word.

Data are shifted into a 4-bit register on the data-bit clock. Enabled by the "4th bit" flag, these 4 data bits are loaded in parallel into a 3- by 4-bit shift register. The 4-bit register and the 3- by 4-bit shift register hold a 16-bit word that is unloaded into a 5- by 16-bit shift register at the end of every 16th bit in the 80-bit data word.

The first 4 bits in an 84-bit word are decoded when the first 4 have entered the 4-bit shift register. These 4 bits are used to determine whether an alert flipflop EV should be set after the following 80 bits have been loaded into the 5- by 16 buffer. The load control unit must accept the 80 bits of data within 20 bit times (the time until the first 16 bits from the next edge data or command words must be loaded into the 5- by 16-bit buffer).

If a null command is detected, EV will not be set and the data entering the 80-bit buffer will not be sent on to the load control unit.

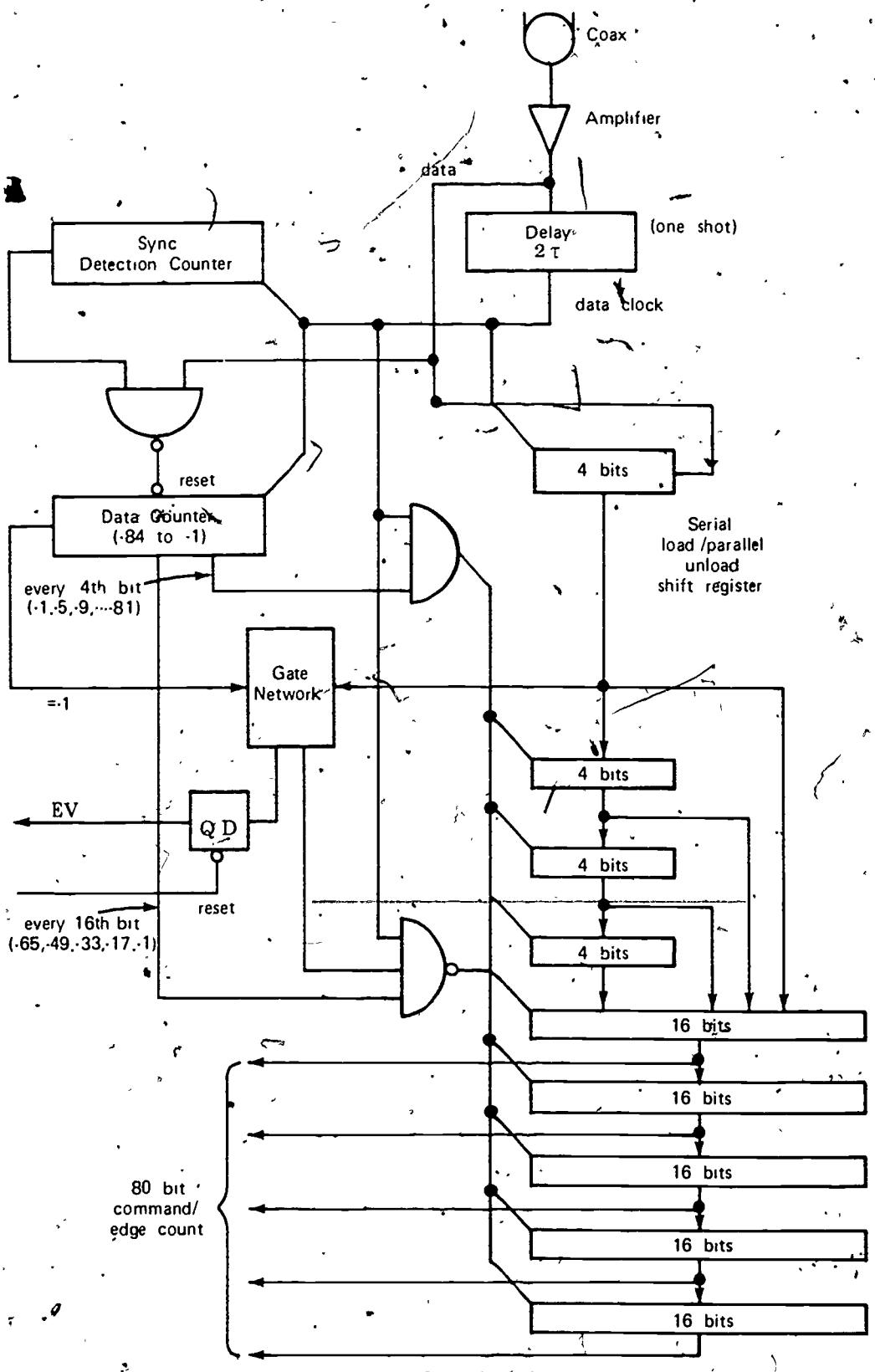


Figure 33 – Coax Receiver

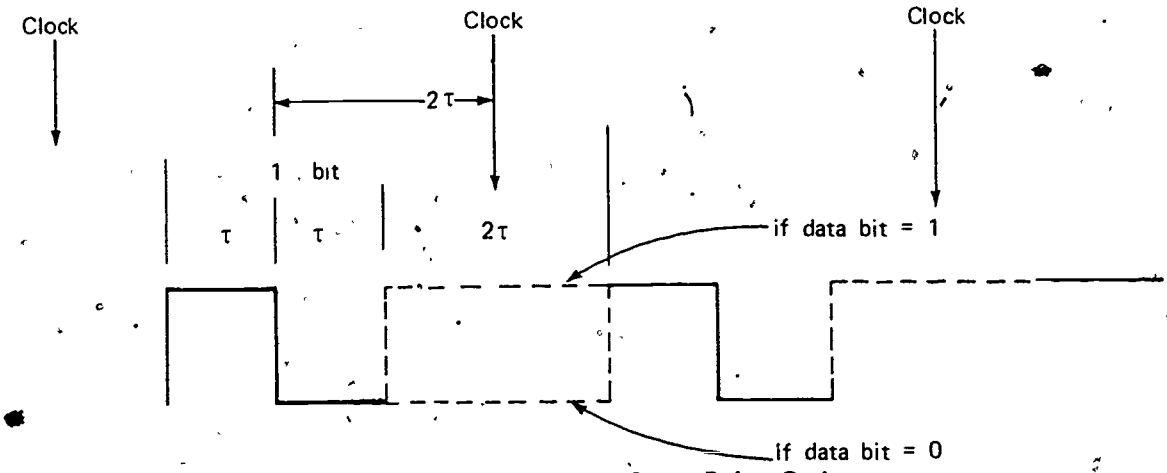


Figure 34 – Coax Pulse Code

For a 100 $\cdot 10^6$-bit-per-second transmission rate, $\tau = 2.5$ nanoseconds. The "front end" of the receiver described above (network feeding the 5- by 16-bit buffer) is able to run up to 6.10^6 baud, limited by the one-shot FF acting as a 2τ delay. Replacing that device with a delay line permits the circuit to run up to 30.10^6 baud. In order to achieve 100.10^6 baud, ECL devices must be employed, although the approach is similar.

Introduction to Terminal Commands

The terminal has a Layer Address Register (TLAR) but not a Bank Address Register (TBAR) since all banks are clocked (read) in unison when an image is decoded.

During memory updating by the communication controller, banks may be loaded individually as well as a layer at a time (the latter in cases where memory is being cleared). Thus, the controller contains a register TBAR serving as a terminal's bank address register during loading. Together, during loading TLAR in a terminal and TBAR in the controller point to the memory location that will be altered if the appropriate control lines are activated with an edge word on the terminals' input data bus.

Command words appear as in Figure 35. There are basically three types of commands: (a) those that begin or end a load sequence and/or seek absolute or relative addresses and/or clear a portion of memory; (b) those that write an edge word or skip by a single memory location; and (c) a command that loads a "look up" RAM in the decoder portion of the terminal.

"Address seek" commands have a Layer Address (LA), Bank Address (BA) and a layer relative address, DA. The communication controller enables clocks to a layer of memory and places LA on the terminal's input data bus so that the terminal can inhibit

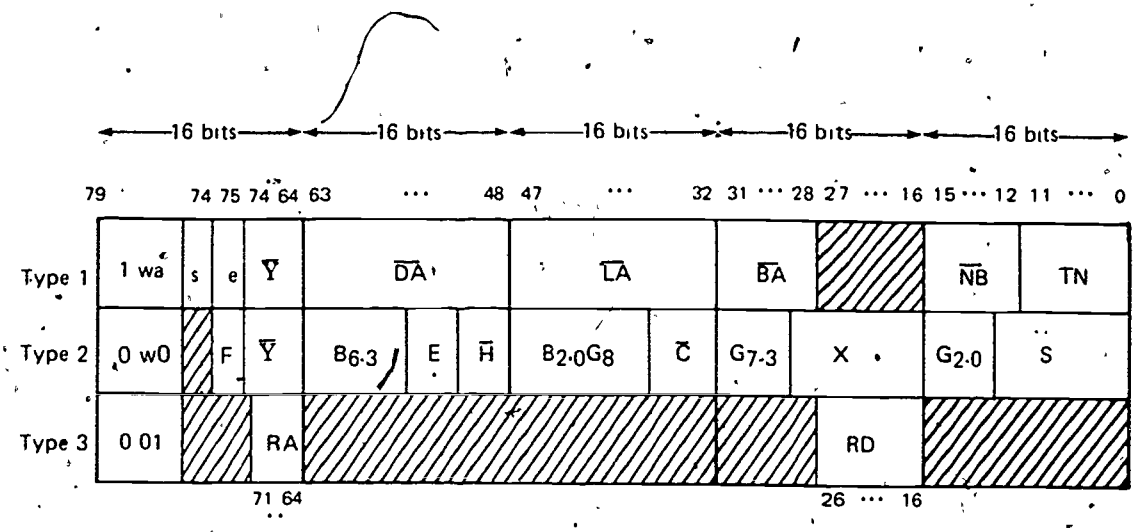


Figure 35 – General Command Words

the advancing of memory if TLA (contents of its TLAR) equals LA during the DA clock periods when memory is advanced a layer at a time.

Thus, if relative addressing is desired, LA is made an impossible address, DA serving as the relative address. If absolute addressing is desired, DA is made large enough so that TLA reaches LA before DA counts down.

This technique eliminates, during loading, the need to receive data from any terminal, thereby simplifying the communication controller. Also, it permits seeking an address within a dynamic shift register memory system which must be clocked at least every 100 microseconds, yet whose present memory address is unknown. That is, to give a command to seek an address can require over 200 microseconds before TLA=LA, yet, depending upon the initial value of TLA, LA may be reached immediately, leaving 200 microseconds until memory is clocked again.

Since the central system may be located miles away on the other end of a coax, there is no time to relay back to it that the terminal has reached the desired address. Thus, the central system must wait the worst case delay which is around 200 microseconds, a time which is short compared to the 5 milliseconds required to transmit 5,000 edges over a 100.10^6 baud line, but large compared to the maximum memory clock interval of 100 microseconds.

In order to hold the memory clock interval to under 100 microseconds on seek commands where TLA is not known by the central system, a sequence of n (3 or 4) address seek commands must be utilized, each employing a DA of under 750 (750 x 134 nanoseconds = ∿ 100 microseconds), the nth command addressing LA, the n-1th command addressing LA-1—the first command addressing LA - n + 1.

Obviously, once TLA is known, commands with any DA are permitted, the central system timing itself (and sending null words down the coax) to guarantee the terminal will reach the desired address just ahead of the next data command.

The attention command uses an argument Terminal Name (TN) to put terminal TN into the load mode, the mode where its memory is under control of the communication controller. Number of Banks (NB) is an argument that is necessary to tell the controller how many banks of memory terminal TN has (in cases where N—number of banks) varies among terminals. NB is used by the controller to cycle TBAR through the N bank address (addresses $0 \rightarrow N-1 = NB$) for banks 1 through N.

To simplify edge data transmission and/or the communications controller, the "attention" and "end attention" commands respectively unstagger and restagger the data in a terminal. Bits e and s in these commands are used for this purpose. The variable e, if set, forces a stagger or unstagger (specified by s) to be done after DA memory loadings. DA must be greater than 258 for unstaggering, since the smaller section of memory may not be aligned with the larger section of memory when the terminal enters load mode.

The variable w, if set, causes edge or null data to enter memory as it is clocked. For instance, clear memory commands are no more than seek commands that alter memory as it is shifted.

Detailed Command Descriptions

Figure 36 illustrates the detailed command descriptions that are discussed in this section.

Attention, New Image (ANI). This command puts terminal TN into load mode and specified NB (one less than the number of banks) for that terminal. This command zeros TLAR and resets the alignment counter. In load mode and with the above counter reset, memory will act as a 1282 N-word shift register.

Following this command, 1282 x N edges or nulls must be written, obeying the blocking data format.

This command does not have to unstagger memory because the memory's old contents are not saved.

Attention, Modify Image (AMI). This command puts terminal TN in load mode and resets its TBAR to zero. It does not reset the realignment counter since portions of the memory contents must not be altered or reordered.

DA of this command must exceed 257 to guarantee the small section of memory is clocked around until alignment occurs and the larger memory loop is formed.

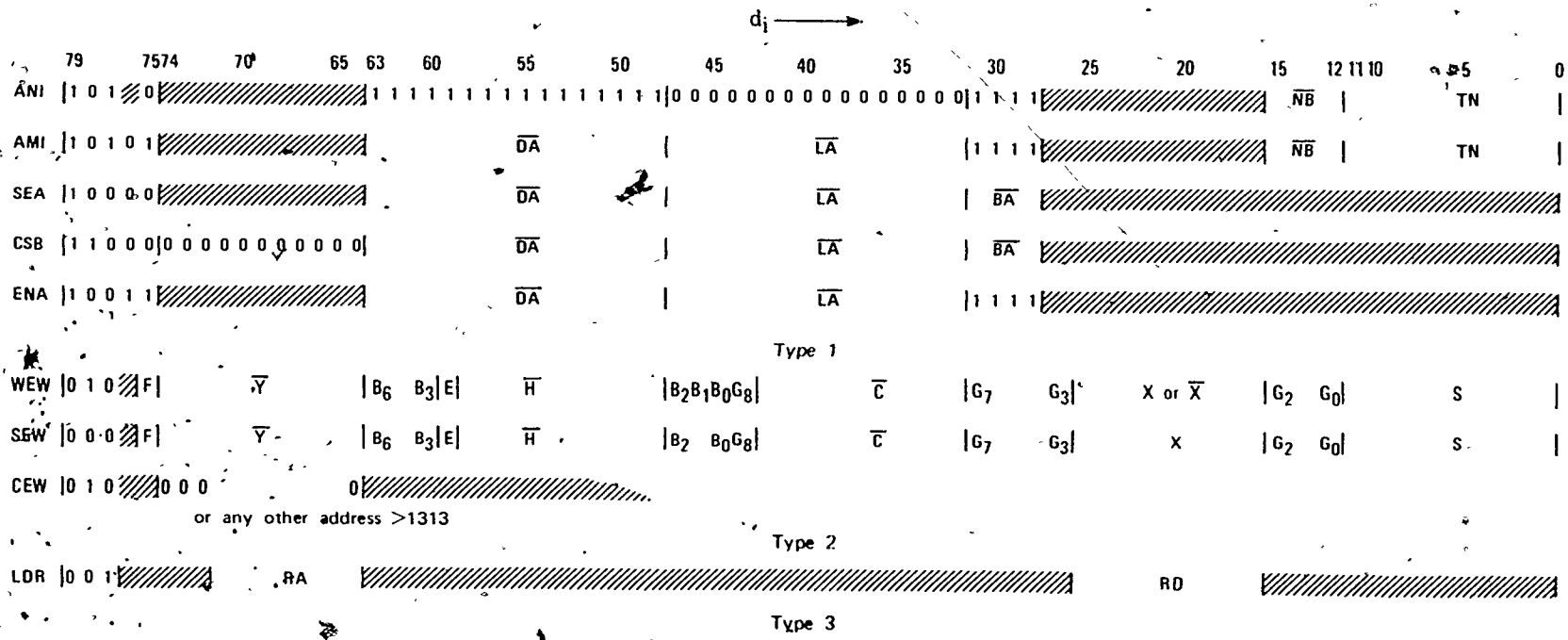


Figure 36 – **Detailed Command Formats**

Following alignment, memory operates as a 1282 N-word shift register, with TLAR incrementing along with memory. The TLA and LA comparison is inhibited until alignment occurs, so that a chance TLA = LA before alignment will not inhibit the clocking of the small section of memory.

BA = 0 because unstaggering requires that TBA = 0.

After an AMI command, SEA (SEek Address) commands must follow for reasons indicated in a preceding section.

SEek Address (SEA). This command moves memory forward until TBA = BA and either TLA = LA or TLA is incremented DA times.

If the initial contents of TBAR are not 0 and DA $\neq$ 0, memory is clocked one bank at a time (starting at bank TBA + 1), (TBA = 0 through N-1 for banks 1 through N respectively) until TBA = 0. Then all N banks are clocked together until TLA = LA or TLA has advanced by DA (module 1282).

After DA layer clock enables (the enables following TLA = LA being ignored by the terminal), the banks are clocked individually until TBA = BA.

If DA = 0, BA must be greater than or equal to TBA for a meaningful result. DA is not constrained in this command when TLA is known (because of a previous command) since the central system can predict accurately when to send the next command which must be within 100 microseconds from the time the terminal actually reaches the last address of the last command. Thus, only the SEA commands just following an AMI command must have DA $\leq$ 750.

Clear Sub-Block (CSB). This command differs from an SEA command in that it writes in "null" edge data at all addresses between the starting address (contents of TLAR and TBAR) up to but not including the address of the command LA, BA.

ENd Attention (ENA). This command is permitted to do a seek to an address with BA $\neq$ 0. After reaching this address, memory is staggered and the terminal taken out of load mode (i.e., put back into display mode).

Write Edge Word (WEW). This command stores an edge word at the address TLA, TBA and increments the address by 1 (TBA is incremented by 1 modulo NB + 1, and TLA is incremented by one (module 1282) when TBA advances to zero).

Clear Edge Word (CEW). Same as WEW command, except a null edge is written in, that is, an edge that will never be detected as crossing scan lines 1 through 1313.

Skip Edge Word (SEW). Same as WEW command, except memory contents are not altered, that is, address incremented by 1 is its only effect.

Terminal Load Control Unit

This portion of the communication controller accepts an 80-bit command edge data word from the coax receiver and initiates a decode cycle whose length varies from 2 clock periods (each 134 nanoseconds) to more than DA clock periods (for address seeking commands).

Referring to Figure 37, when EV is sensed "high" on a 67 clock, a VA flipflop is set which resets EV and enables the strobing of the 80-bit input into an 80-bit bus register. This 80-bit register has a portion that can act also as a counter where DA can be decremented ($\overline{DA}$ incremented).

The 80-bit bus register outputs directly to the input data bus for all terminals served by a single communication controller (up to 2,000 terminals when intermediate bus driver gates are added).

When this register is loaded with bits $d_{79} = d_{77} = 1$, (attention commands) the NB bits are also loaded into the 4-bit Number of Banks Register (NBR) and TBAR is set to zero. Whenever TBA = NB, the next state of TBAR is zero (i.e., TBA is incremented by one modulo NB + 1).

A load control network decodes the upper 5 bits of the command in the 80-bit bus register and generates a sequence of control or clock enables to the terminals. Most of the control signals are ignored by a terminal until it is put in the load mode. In particular, an attention command in the bus register causes Terminal Address Enable (TAE) to go true for one clock period (134 nanoseconds). All terminals look at TN, which has been placed upon the input bus. Only terminal TN sets its LD flipflop, thereby entering the load mode. Only when a terminal has its LD flipflop set, does it respond to the remaining control lines.

If the ANI command were given, Reset Counter (RSCN) control line is raised (when TAE falls) so that the addressed terminal will reset its TLAR and its alignment counter. This alignment counter is reset so that the large 1282 memory loop will be formed independent of the state that the smaller section of memory was in.

When the terminal enters load mode, the clock enable lines to its N banks are switched over to the clock enable lines from the terminal load control unit, lines ERCI through ERCN.

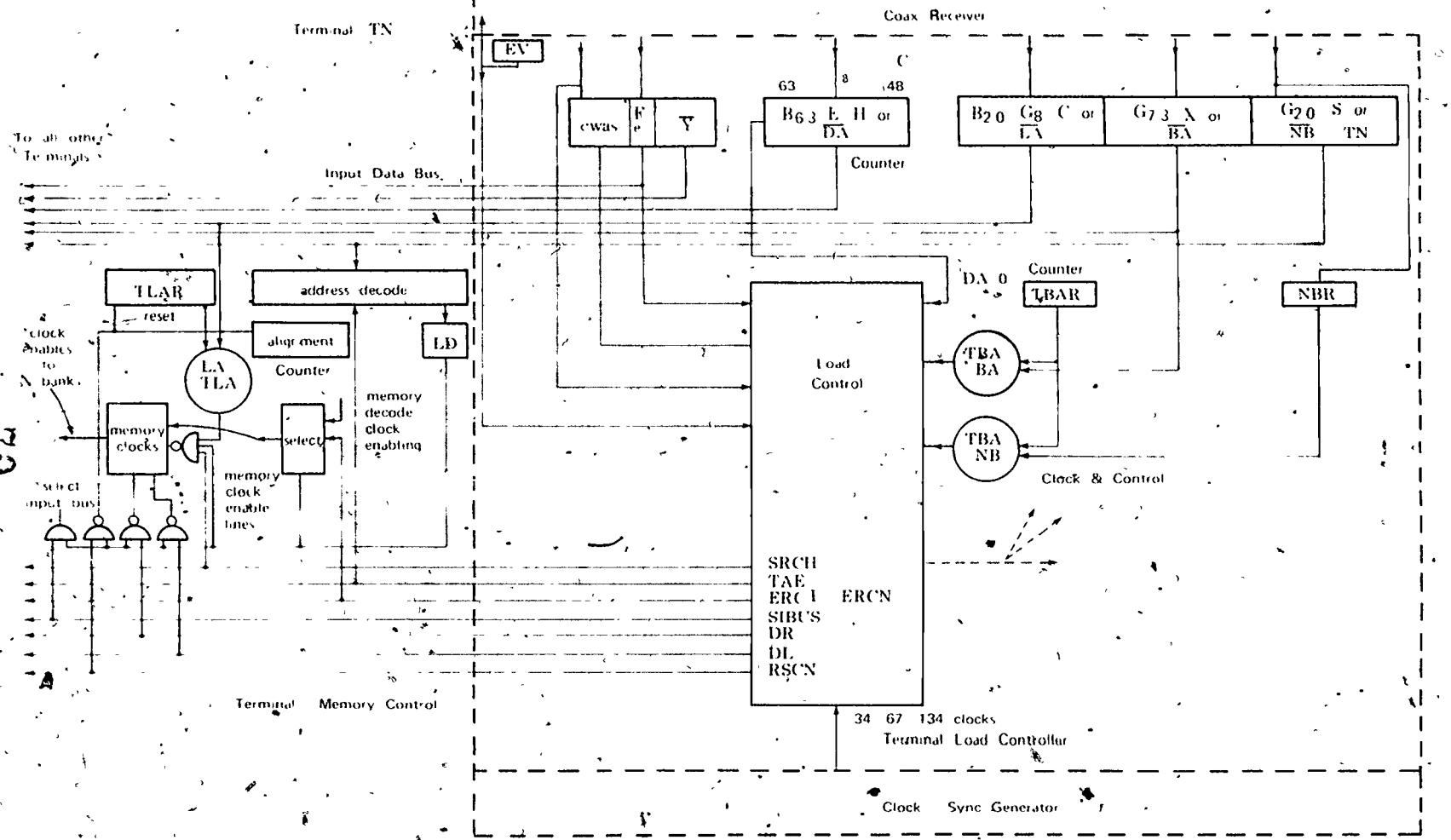


Figure 37 – **Terminal Load Control**

During address seeking, either one or all of these lines are raised. Whenever bank N enable line is raised, TLAR in the terminal increments along with the clocking of the Nth bank (or all N banks). This is the same time that TBAR resets to zero in the controller.

The control signal SRCH is activated in address seek command until DA is decremented to zero. The terminal requires SRCH to be "true" before it will let the condition LA = TLA inhibit clocking of memory even though ERCI → ERCN are true.

In address seek commands DA in the bus register is decremented every time all N banks are clocked.

A control signal SIBUS is "true" if the command has w = 1 (command bit d_{78}). This SIBUS = 1 forces the addressed terminal to accept data into its memory input registers from the input bus rather than from a shift register. Thus, memory will be written into as memory is clocked.

At the end of an AMI command when DA has decremented to 0 a control line DL is raised for *two* clock periods and all N clock enable lines (ERCn) are raised for the first of those two clock periods. This sequence clocks only the right 52-bit portion of all the 76-bit-wide banks of memory. The effect of this sequence is to unstagger memory. DL must be held high two units of time because the disabling of the left 24 bits is done on the actual clock to the banks, not its enables. This clock extends in time into the period following its enable. Because the alignment counter is probably not reset at the beginning of an AMI command, DA must exceed 257.

An End Attention command (ENA) acts as a seek command to an address with BA = 0. Because e = s = 1 in the control bits of this command, memory will be staggered after the address is reached. This occurs when DR is held true for two units of time with ERCI through ERCN held high for the first of those periods. DR disables the clock to the right 52 bits of all N banks so that staggering results.

The terminal, upon receiving DR = 1 and ERCN = 0, resets its Load (LD) flipflop, ending the load mode.

For WEW, CEW, and SEW commands, only one of the ERCN lines are raised, causing (a) memory to advance one edge word, and (b) the edge word on the input bus to be written into only one bank (for a WEW or a CEW command only).

A Clear-Sub-Block Command (CSB) acts as a SEA command except SIBUS is held high during the enabling of the ERCI through ERCN lines.

TIMING GENERATOR

This unit generates for each scanline pulses BG, BG-1, BGE, HRP, SRC, and SRC + 1, used by the terminals for timing purposes (see Figure 38).

Also generated is the y value (y_{10}, y_9, . . . , y_0) of the scanline, which increments by five between scanline decode periods or which resets to an integer between 1 and 5 if an end of field (vertical scan) is detected. Figure 38 shows the sequence of y values for each field.

Field	y ⟶	# Scanlines
1	1, 6, 11, • • • 1311	262
2	3, 8, 13 • • • 1313	262
3	5, 10, 15 • • • 1310	261
4	2, 7, 12 • • • 1312	262
5	4, 9, 14 • • • 1309	261

Figure 38 – Interlace Specifications

Also generated are horizontal and vertical sync (HSYNC and VSYNC), a 67-nanosecond clock and a 134-nanosecond clock enable,

The timing signals are derived from several counters. One counter counts to 98 in 98 x 134 nanoseconds signaling the end of a 1/5 scanline period and enabling a second and third counter. The second counter counts to 5, signaling the end of a scanline. The third counter counts to 1313, signaling the end of a vertical trace period (field). Because 1315/5 is 262 3/5, a 5 to 1 interlace results. Horizontal sync is derived from the first and second counter. Vertical sync is derived from the first and third counter.

A fourth "field counter" is employed which holds the initial value for y at the beginning of a field and which is used to reset the y counter between fields. Because three fields consist of 262 scanlines and two fields consist of 261 scanlines, a fifth counter is employed to (a) count to 261 or 262, and (b) enable the resetting of the y register when end counts are reached. This counter increments at the end of the fourth fifth of a scanline (near end of visible scanline period).